Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Development of a practical LoRaWAN scenario

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Klever Felipe Robles Hidalgo

ADVISOR: Carles Gómez Montenegro, Rafael Vidal Ferré

DATE: February, 7th 2019

Title: Development of a practical LoRaWAN scenario

Author: Klever Felipe Robles Hidalgo

Advisor: Carles Gómez Montenegro, Rafael Vidal Ferré

Date: February, 7th 2019

Abstract

LoRaWAN (Long Range Wide Area Network) is a networking solution for Internet of Things (IoT) devices offering long range, low power, and low bit rate. In this technology, end-devices use LoRa to physically communicate with gateways.

The aim of this master thesis is to develop a practical LoRaWAN scenario focused on educational purposes in order to help students understand the LoRa and LoRaWAN basic parameters and mechanisms such as activation procedures, data rates, transmit power and link quality. The processes to obtain the LoRa packets in a practical LoRaWAN scenario and the tools to analyse them are also presented in this project.

The report is dived in four chapters: The first chapter covers a brief overview of the LoRaWAN technology and how it works. The second chapter is focused on all the software and hardware tools and devices that are necessary to develop the practical scenario. The third chapter is a complete guide of the development of the network. The final chapter analyses the different possibilities and parameters that can be managed along with a guide for a Lab session where the steps, for the different class roles (i.e. professor, student), to develop a LoRaWAN practical environment, are presented.

CONTENTS

TABLE INDEX

FIGURES INDEX

INTRODUCTION

The Internet of Things (IoT) term has been used to indicate different technologies whose main object is to enable the physical objects to connect to the Internet. Some of the most popular technologies associated with IoT are radio frequency identifiers, and short-range wireless communication technologies, such as IEEE 802.15.4 and Bluetooth Low Energy (BLE). The majority of these technologies are characterized by their low power consumption and short range, which limits the use cases to limited coverage areas.

Next generation cellular networks plan to deal with the challenge of ubiquitous coverage of IoT nodes, the current cellular networks were not conceived to provide services to a massive number of devices. 5G cellular networks are expected to natively support IoT connectivity. Meanwhile, the IoT market keeps growing and this demand needs to be taken care of, many commercial technologies have emerged based on a paradigm referred to as Low Power Wide Area Network (LPWAN). LPWAN networks are characterized by long range links, low power consumption and a capacity to connect a high number of IoT devices.

One of the most popular LPWAN technologies is LoRa, which is a long-range wireless communications system, promoted by the LoRa Alliance. LoRa refers to two distinct layers: a physical layer using the Chirp Spread Spectrum (CSS) radio modulation technique and a MAC layer protocol (LoRaWAN).

In this work, the development of a practical LoRaWAN scenario is made, where the majority of the LoRaWAN network parameters can be analysed and managed such as: payload formats, configuration of end-devices, different data rates, transmission power, among other parameters. All of this is intended to be introduced in an educational environment.

The structure of this document is as it follows:

The first chapter is focused on theoretical concepts that have to be covered in order to have a complete idea of the purposed LoRaWAN scenario. In the first part, a brief introduction to the LPWAN technologies is made. The second part is the working principles of LoRa and LoRaWAN covering the network architecture, the payload format, the MAC commands, the key properties and the different device activation procedures.

The second chapter covers the software and hardware needed in order to develop the scenario. It describes their key features. Also, the chapter shows a comparison between two fundamental software alternatives in order to develop a LoRaWAN network.

The third chapter presents the complete development of the practical scenario in a type of guide way so students can replicate and analyse the different LoRa/LoRaWAN parameters. This chapter covers activation of the gateways, activation of the end-devices and creation of a private back-end.

The fourth chapter contains the guides and explanation of the possibilities that can be exploited in an educational environment in order to understand the different parts of a LoRaWAN environment such as: LoRaWAN key concepts, packet formats, data rates and transmission power.

The final part covers the conclusions of the work, future lines of development, sustainability considerations and ethical considerations. At the end of this work, annexes contain the different scripts used for the development of chapter 3 and 4.

CHAPTER 1. LORA/LORAWAN

This chapter covers a brief introduction to IoT, and the different wireless technologies used in this field, and a more in-depth view of the LoRa/LoRaWAN technology is given, covering the LoRa physical layer and the LoRaWAN MAC protocol.

1.1. Introduction

It is estimated that by 2020 50 billion of devices will be connected to the Internet [1]. The term IoT has been used for different kinds of technologies that in general their final scope is to connect physical objects to the Internet [2]. The essential difference between “traditional Internet” and Internet of Things is that in IoT the devices have less resources available such as energy, bandwidth, memory and less power processing. This is mainly because of two reasons either the devices are battery driven, and maximizing lifetime is a priority or because as mentioned before the number of devices is expected to be very high in the future [3].

The constraints mentioned limit the applicability of traditional wireless networks, as well as of technologies. Due to this new protocols and technologies such as Low Power Wide Area Network (LPWAN) have emerged [3]. This family of technologies allows the deployment of a wide area network with gateways and, in some cases, to adapt parameters such as data rate, power, and modulation among others. This way, the end-devices have an efficient power consumption and a link range in the order of kilometers, which reduces the amount of infrastructure needed for connecting the devices.

1.2. IoT Communication Technologies

The development of the IoT is an extremely challenging topic, and the debate on how to put it into practice is still open. The discussion involves all layers of the protocol stack, from physical transmission up to data representation and service composition. However, the whole IoT system rests on the wireless technologies that are used to provide access to the end-devices [4].

1.2.1. Short-Range IoT Communications

Over the years, multihop short-range transmission technologies have been considered a viable way to implement IoT networks. These communication standards are characterized by very low power consumption, which is an essential requirement for IoT devices, but a limited coverage area, creating a major issue when trying to deploy on a wide coverage scenario (for example, smart city applications).

This section provides a brief overview of the most common communication technologies used in IoT.

IEEE 802.15.4 is a standard that defines the physical and data link layer for Low-Rate Wireless Personal Area Networks (LR-WPANs). It supports three un-licensed frequency bands:

- Europe 868 MHz
- North America 928 MHz
- Worldwide 2.4 GHz

It offers data rates up to 250 kbps and a distance with a clear line-of-sight up to of up to a few tens of meters [5].

Bluetooth Low Energy (BLE) was originally created to replace wired connections between devices, such as mobile phones, laptops, audio devices, keyboards among others. It offers a data rate up to 1 Mbps and a short range of up to a few tens of meters, with low power consumption [3].
After several revisions Bluetooth 5.0 was presented in 2016, principally aimed at IoT. It offers bandwidth up to 2 Mbps, low energy consumption and with range coverage up to 240 m [6].

1.2.2. Low Power Wide Area Networks

Low Power Wide Area Networks offer features such as wide-area connectivity for low power and low data rate devices, not provided by legacy wireless technologies [8].

In order to design a LPWAN there are some common requirements:

- Requires a minimum energy consumption in order to support a long lifetime with simple batteries.
- Cost is an important factor, especially in the end-devices, they must be easy to install and of low cost.
- The network infrastructure has to be easy to setup.
- Secure transmission between final user and end-device.
- A robust modulation.

This section gives a quick overview of the most prominent technologies for LPWAN: Sigfox, DASH7, Ingenu, NB-IoT and 802.11 ah. LoRa technology will be explained in greater detail in section 1.3.

Sigfox is a variation of the cellular system that enables remote devices to connect to Ultra Narrow Band (UNB). In Europe, Sigfox operates on the 868 MHz band. Each end-device can send up to 140 messages per day, with a data rate of up to 100 bps (600 bps in the US). Each access point theoretically can handle up to a million end-devices, with a coverage area of 30-50 km in rural areas and 3-10 km in urban areas [4].

Ingenu is a technology developed by On-Ramp Wireless, works in the 2.4 GHz band but, thanks to a robust physical layer design it can still operate over long-range wireless links. It can cover ranges up to 15 km with a data rate of 8 kb/s [4].

DASH7 is a wireless sensor and actuator full Open Systems Interconnection (OSI) stack protocol that operates in three unlicensed bands, 433 MHz, 868 MHz and 915 MHz. DASH7 provides communication for up to 2 km, low latency, mobility support, multi-layer battery life and a date rate up to 167 kbit/s [3].

NB-IoT Narrowband IoT is a radio technology developed by 3GPP. It has been designed to offer extended coverage compared to the traditional GSM networks. NB-IoT can improve uplink capacity for users in bad coverage areas through single tone transmissions (250 kb/s). New physical layer signals and channels, such as synchronisation signals and physical random access channel, are designed to meet the demanding requirement of extended coverage and low device complexity.

IEEE 802.11 ah operates at sub-1-GHz bands (license-free). It supports a range up to 1 km at the default transmission power of 200 mW. It can operate at 4 Mbps or 7.8 Mbps depending on the bandwidth. Thanks to the new modulation and coding schemes of 802.11ac high data rates can be achieved [7].

1.3. LoRa

LoRa is a long-range wireless communications system, promoted by the LoRa Alliance and patented by Semtech in 2014. It operates on the 433, 868 or 915 MHz bands, depending on the region in which it is deployed, for Europe the band that is used is the 868 MHz band.

LoRa is a chirp spread spectrum (CSS) modulation, which uses frequency chirps with a linear variation of frequency over time in order to encode information.

In order to improve the spectral efficiency and the network capacity, the LoRa modulation presents six orthogonal spreading factors (SF), that result in six different data rates. An increasing value in SF means a higher coverage range but a lower data rate.

The LoRa system distinguishes between uplink and downlink messages.

- **Uplink Messages:** Are messages that are sent by end-devices (nodes) to the Network Server relayed by one or many gateways.
- **Downlink Messages:** Are messages that are sent by the Network Server to only one end-device, which are relayed by a single gateway [9].

1.3.1. LoRa Network Architecture

A typical LoRa network is a “a star-of-stars topology”. It includes three types of devices: End-devices, Gateway and Network Server, as shown in Figure 1.1.

End-devices communicate with gateways using LoRa with LoRaWAN protocol. Gateways forward raw LoRaWAN frames from devices to a network server over a backhaul interface, typically Ethernet or 3G.

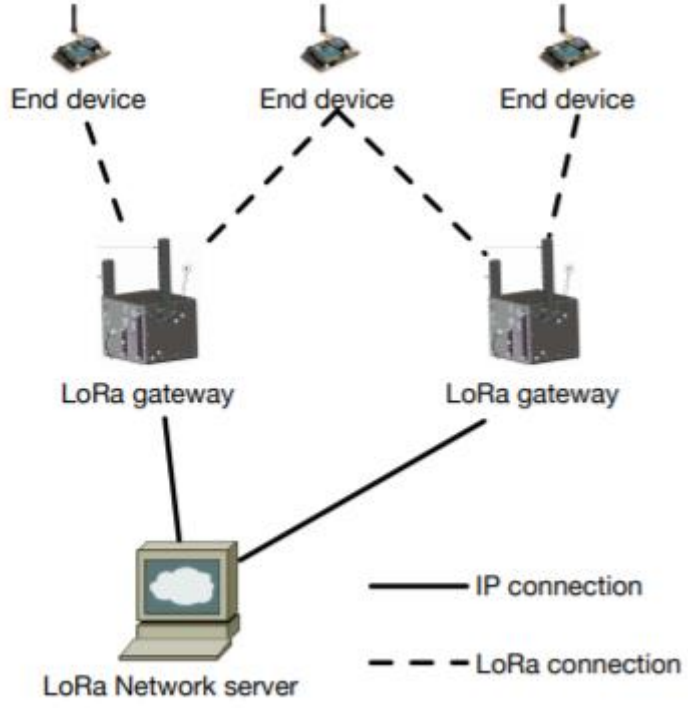

Fig. 1.1 LoRa Network architecture [3].

1.3.2. LoRa Physical Layer Parameters

In order to achieve the best performance in a given scenario, different parameters can be configured: Transmission Power (TP), Carrier Frequency (CF), Bandwidth (BW), Spreading Factor(SF) and Code Rate (CR) [10].

In this section a brief description of the parameters listed is detailed:

- **Transmission Power (TP):** Can vary from -4 dBm to 20 dBm, in 1 dB steps. Due to implementation limits, the actual range is from 2 dBm to 20 dbm.
- **Carrier Frequency (CF):** Can be adjusted from 137 MHz to 1020 MHz in 6 Hz steps. Depending on the LoRa chip it can be limited to 860 MHz to 1020 MHz.
- **Spreading Factors (SF):** In order to improve the spectral efficiency and the network capacity, the LoRa modulation presents six orthogonal spreading factors (SF), that result in six different data rates. An increasing

value in SF means a higher coverage range but a lower data rate. The spreading factor can be selected from 6 to 12.

- **Bandwidth (BW):** BW is the width of the transmission band. LoRa operates with a bandwidth of 125 kHz, 250 kHz or at 500 kHz. With higher BW a lower sensitivity and higher data rate is achieved, with lower BW higher sensitivity and lower data rate is obtained.
- **Coding Rate (CR):** It is the forward error correction used by LoRa, that offers protection against interference. It can be set to 4/5, 4/6 or 4/8. Higher CR means higher protection but lower data rate.

1.3.3. LoRa Key Properties

Some key aspects that make LoRa outstand to other IoT technologies are presented below [11]:

- **Scalable Bandwidth:** LoRa modulation is bandwidth and frequency scalable. Frequency hopping can be used in narrow band and wideband direct sequence applications.
- **High Robustness:** Due to the asynchronous nature a LoRa signal is very resistant to the in-band and out-band interference mechanisms.
- **Low Power Consumption:** The output power of the transmitter can be reduced compared to FSK link while maintaining the same or even a better link budget.
- **Robust against multipath and fading:** Due to the chirp pulse being relatively broadband LoRa offers immunity to multipath and fading, making it ideal for urban and suburban use.
- **Doppler Resistant:** Frequency offset between the transmitter and the receiver are equivalent to timing offsets thanks to the linearity of the chirp.
- **Long Range Capability:** With the same power and throughput the link budget of LoRa exceeds a FSK link.
- **Enhanced Network Capacity:** LoRa enables multiple spread signals to be transmitted at the same time and on the same channel without degradation due to the employment of orthogonal spreading factors.

1.3.4. Physical Layer Frame Format

Although in LoRa modulation arbitrary frames can be transmitted, Semtech has specified a physical frame format in which the bandwidth and the spreading factor are constant for a frame [3] (see Figure 1.2).

The LoRa frame begins with a preamble. The preamble starts with a sequence of upchirps[1] that cover the whole frequency band. The last two upchirps encode the sync word, which is a value that is used to differentiate LoRa networks that use the same frequency bands [3].

An optional header comes after the preamble, it indicates the size of the payload, the code rate used for the end of the transmission and whether a cyclic

[1] Signal in which the frequency increases

redundancy check (CRC) is present at the end of the payload. It also includes a CRC to allow the receiver to discard packets with invalid headers.

After the header the payload is sent, and at the end of the frame payload, an optional CRC is sent.

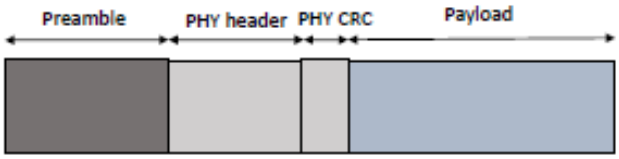

Fig. 1.2 LoRa Frame Structure [12]**.**

1.4. LoRaWAN

LoRaWAN defines a Medium Access Control (MAC) protocol created by the LoRa Alliance that runs on top of LoRa physical layer. It is designed mainly for the communications between multiple end-devices and network gateways.

1.4.1. Components of a LoRaWAN Network

The LoRaWAN network architecture has a star topology, several components of the network are defined in the LoRaWAN specification and are required to form a LoRaWAN network [3]:

- **End-Devices:** Low-power consumption sensors that communicate with gateways using LoRa.
- **Gateways:** Intermediate device responsible for forwarding raw data packets from end nodes towards the network server, over an IP backhaul interface, encapsulating them in UDP/IP packets.
- **Network Server:** Responsible for de-duplicating and decoding the packets sent by the devices and generating the downlink and MAC commands towards end-devices.
- **Application Server:** It can be owned by different third parties. Multiple application layers can be connected to a single network server [12].

The resulting LoRaWAN network architecture can be seen in Figure 1.3.

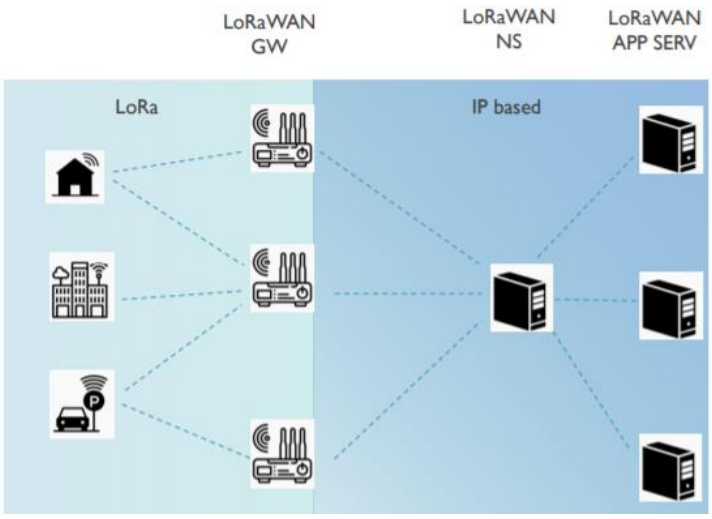

Fig. 1.3 LoRaWAN Network Architecture [12]**.**

1.4.2. LoRaWAN Device Classes

LoRaWAN has three different bidirectional classes of end-devices to address the various needs of applications [12]. The different device classes trade off the network downlink communication latency versus the battery life time as can be summarized in Figure 1.4.

- **Class A:** The class A end-devices, schedule an uplink transmission based on its needs. Each uplink transmission is followed by two short downlink receive windows. Class A devices have the lowest consumption, but also they offer less flexibility on the downlink transmissions. All LoRaWAN end-devices must support class A.
- **Class B:** Class B devices open extra receive windows at scheduled times. A synchronized beacon from the gateway is required to inform the network server when is the end-device listening.
- **Class C:** Class C end-devices have almost continuous receive windows meaning that a device in this class is the one with the more power consumption of the three classes.

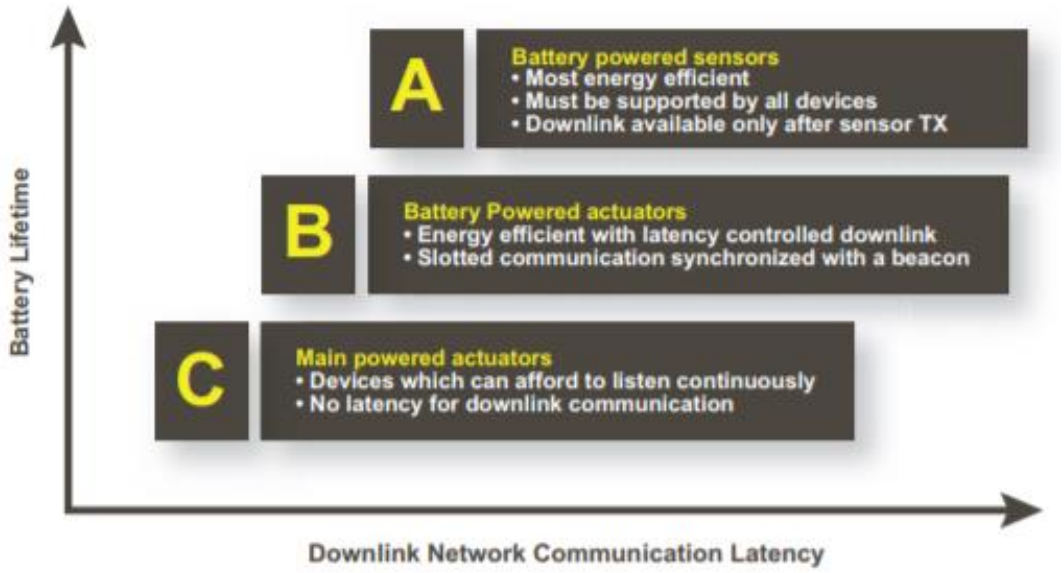

Fig. 1.4 LoRaWAN Network Architecture [13]**.**

1.4.3. LoRaWAN MAC Message Format

All LoRa uplink and downlink messages carry a PHY payload (payload) starting with a single-octet MAC header (MHDR), followed by a MAC payload (MACPayload) and ending with a 4-octet message integrity code (MIC). The message format is detailed in Figure 1.5.

In Table 1.1 a terminology description of the different part of the LoRaWAN MAC message format is shown.

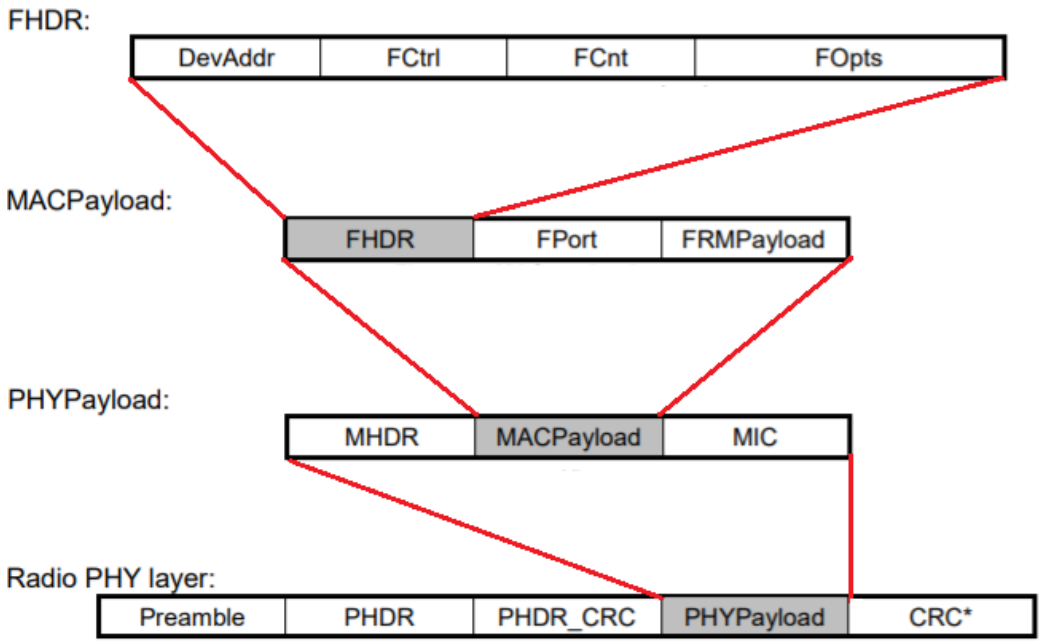

Fig. 1.5 LoRaWAN MAC Message Format [9]**.**

Table 1.1. Model Table

LoRaWAN MAC Messages	Description
MHDR	MAC Header
MAC Payload	Data from the MAC Payload
MIC	Message Integrity Check
FHDR	Frame Header
FPort	Multiplexing Port Field
FRMPayload	Frame Payload
Devaddr	Device Address
FCtrl	Field Control
FCnt	Frame Counter
FOpts	Frame Options

MAC Header

The MAC Header specifies the Message type (Mtype) and, depending on the version, the codification to be used. LoRaWAN distinguishes between six different MAC message types that can be shown in Table 1.2.

Table 1.2. MAC Message Types

MType Value	Description
000	Join Request
001	Join Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	RFU
111	Proprietary

1.4.4. Adaptive Data Rate

An Adaptive Data Rate (ADR) mechanism is built into LoRaWAN for managing the end-device's link parameters in order to increase the packet delivery ratio. It manages the data rate and transmit power of end-devices. When ADR is enabled in the Control Field (FCtrl), the network will optimize the transmissions using the fastest data rate possible.

If the ADR bit is set, the network will control the data rate of the end-device through the appropriate MAC commands. If the ADR bit is not set, the network

will not attempt to control the data rate of the end-device regardless of the received signal quality.

The ADR algorithm used for this project is The Things Network (TTN), that is one of the largest LoRaWAN networks, algorithm which is based on Semtechs ADR algorithm. In Figure 1.6 TTN's ADR algorithm for the uplink part is shown and in Figure 1.7 the downlink part and the ADR response is shown.

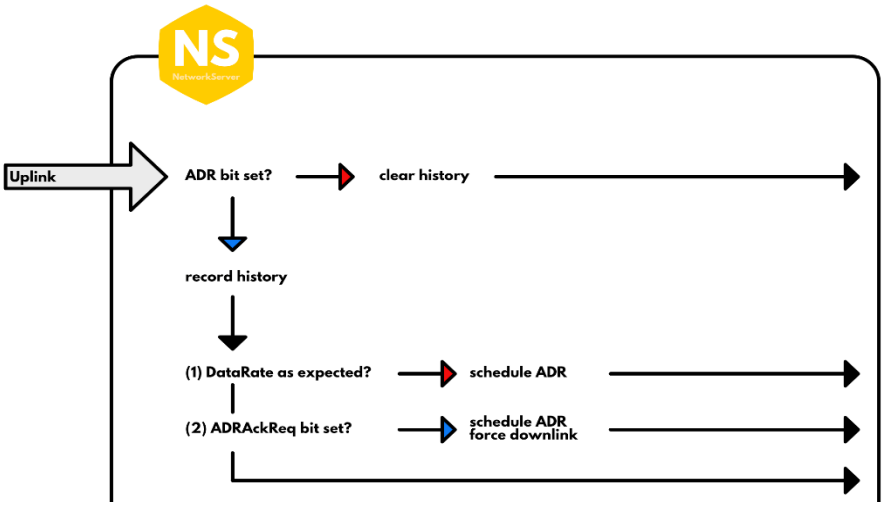

Fig. 1.6 TTN's ADR for uplink.

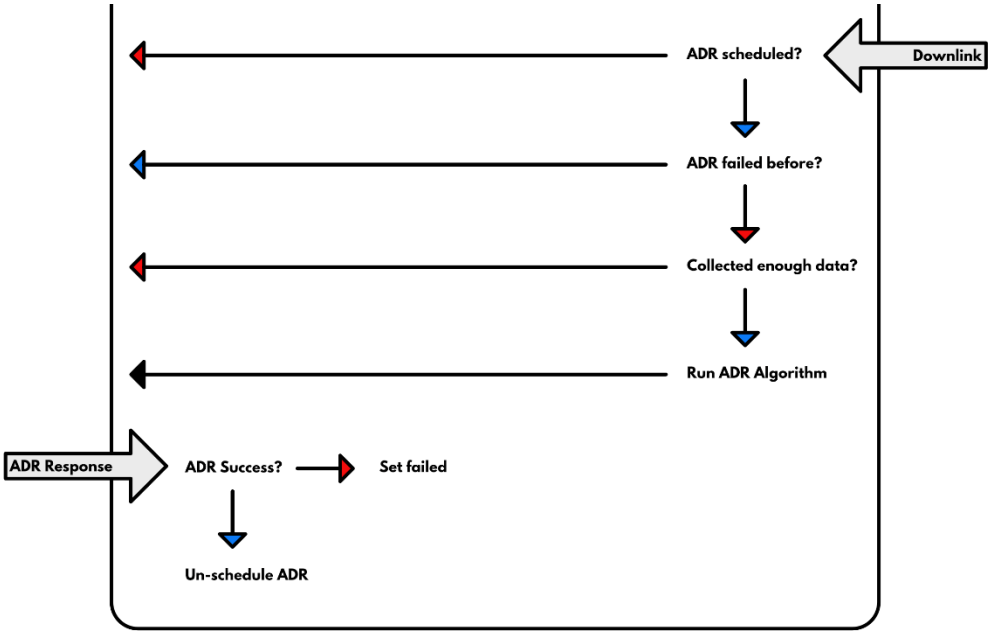

Fig. 1.7 TTN's ADR for downlink.

1.4.5. Activation of End-Devices

In order to join a LoRaWAN network, an end-device must be activated. The activation can be done in two different ways: Over-The-Air Activation (OTAA) and Activation By Personalization (ABP) [3].

Activation parameters

The parameters that an end-device should have in the activation process can be shown in the following table:

Table 1.3. Activation Parameters

Parameter	Description
End-device address (DevAddr)	Consists in 32 bits that identify the end-device in the network.
Application identifier (AppEUI)	Global application ID that identifies the owner of the end-device.
Network session key (NwkSKey)	Used by the Network Server and the end-device to calculate and verify the MIC. It is also used to encrypt and decrypt the MAC payload field.
Application session key (AppSKey)	Used by the network server and the end-device to encrypt and decrypt the payload field of application-specific data messages.

- **Over-The-Air-Activation (OTAA):** An end-device is personalized with keys and identifiers that are saved before the procedure starts. The end node sends join requests. The network server will reply with a join-accept message if the device is permitted to join the network. Next a Rekey indication command, used to confirm security key update and to negotiate the LoRaWAN protocol version running between the end-device and the NS, is sent. Finally the NS sends a Rekey confirmation command (RekeyConf) with the LoRaWAN version supported [9]. The whole process for the OTAA activation can be seen in Fig 1.8.

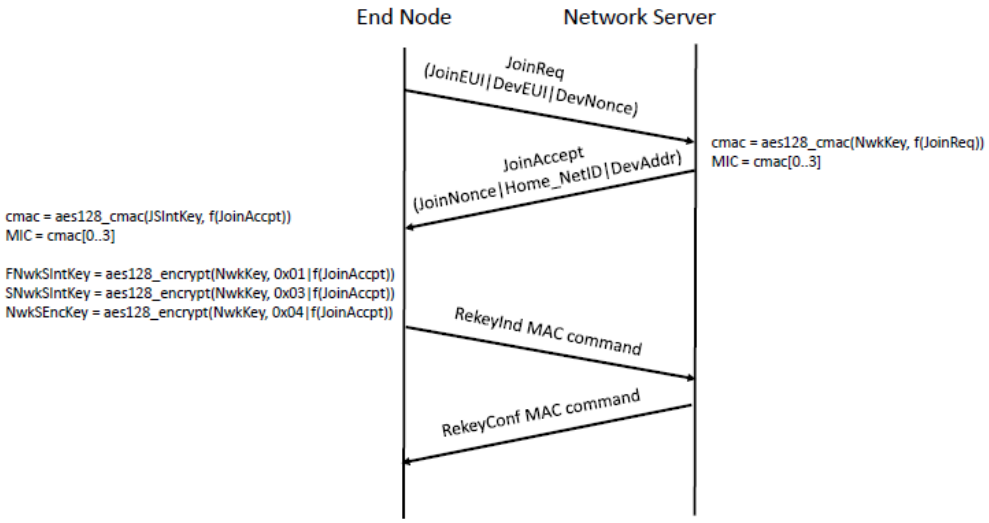

Fig. 1. 8 OTTA process [12].

- **Activation by Personalization (ABP):** The ABP process directly activates the end-device to a specific network omitting the Join-request – Join-accept procedure, which is needed in OTAA.

  All the session keys are stored in the end-device beforehand, instead of being derived from the join procedure.

1.4.6. Data Rate and Transmission Power

In Europe, duty cycles are regulated by section 7.2.3 of the ETSI EN300.220 standard. This standard defines the following sub-bands and their duty cycles [14]:

- g (863.0 – 868.0 MHz): 1%
- g1 (868.0 – 868.6 MHz): 1%
- g2 (868.7 – 869.2 MHz): 0.1%
- g3 (869.4 – 869.65 MHz): 10%
- g4 (869.7 – 870.0 MHz): 1%

Furthermore, each network server operator is free to select additional parameters that have to be compliant with the governmental and LoRaWAN limits. One case is the Frequency Plans for TTN which states beyond duty cycle regulations policies:

TTN Fair Access Policy limits the data each end-device can send, by allowing:

- An average of 30 seconds uplink time on air, per day, per device.
- At most 10 downlink messages per day, including the ACKs for confirmed uplinks [15].

In table 1.4, the Data Rate (DR) and the bit rate achieved can be seen. In table 1.5 the end-device output power (TXPower) is shown. Both tables are for the European frequency band.

Table 1.4. Data Rate [9]

Data Rate	Configuration	Physical bit rate (bits/s)
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 250 kHz	11000
7	FSK: 50 kbps	50000

Table 1.5. Output Power [9]

TxPower	Configuration
0	20 dBm (if supported)
1	14 dBm
2	11 dBm
3	8 dBm
4	5 dBm
5	2 dBm
6….15	RFU

CHAPTER 2. EXPERIMENTAL ENVIRONMENT

In this chapter, the software and hardware employed in order to deploy the LoRaWAN Network for this project will be exposed.

2.1. Hardware

In order to deploy a LoRaWAN network, the following hardware equipment is needed:

- LoRaWAN Module: Device in charge of sending (or receiving) the data to (from) the gateway.
- LoRaWAN Gateway: Device in charge of forwarding the traffic from the end-devices.

2.1.1. LoRaWAN End Device Module

The module used for the development of this project is a Dragino LoRa Shield [16] v1.4 (Figure 2.1) mounted on an Arduino Uno.

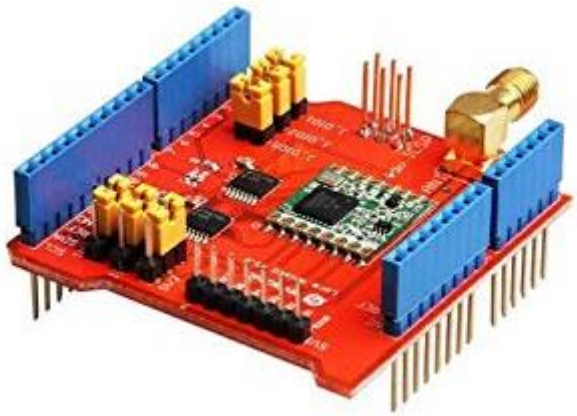

Fig. 2.1 Dragino v1.4 [16]

The Dragino Shield is a long range transceiver on an Arduino shield form factor based on Semtech SX1278 chip. The principal device features are:

- Works on the 868 MHz Frequency band.
- 168 dB maximum link budget.
- FSK, GFSK, MSK, GMSK, LoRaTM and OOK modulation
- Low Power consumption
- +20 dBm - 100 mW constant RF output vs. voltage supply.
- +14 dBm high efficiency PA.

- Programmable bit rate up to 300 kbps.
- High sensitivity: down to -148 dBm.
- Bullet-proof front end: IIP3 = -12.5 dBm.
- Excellent blocking immunity.
- Low RX current of 10.3 mA, 200 nA register retention.
- Fully integrated synthesizer with a resolution of 61 Hz.
- Built-in temperature sensor and low battery indicator.
- Built-in bit synchronizer for clock recovery.
- Preamble detection.
- 127 dB Dynamic Range RSSI.
- Automatic RF Sense and CAD with ultra-fast AFC.
- Packet engine up to 256 bytes with CRC.

2.1.2. Gateway

The gateway used for this project is a The Things Gateway (Fig 2.2).

Fig. 2.2 The Things Gateway [17]

The Things Gateway is a device created by TTN, which allows low-power devices to connect to a decentralized network with a range up to 10 km. In the TTN architecture, gateways are used as a bridge between the LoRaWAN network and the Internet [18].

The Gateway works on the EU frequency band 868 MHz using a LoRa gateway card, the LG8271 from Microchip (Fig 2.3), which combines eight receiving channels and one transmit channel.

Fig. 2.3 The Things Gateway [19]

The main processor of The Things Gateway is a PIC32MZ2048EFM144[2] running at 200 MHz. Communications between the LoRa card and the main processor is done via UART[3]. There are two types of Internet Connection: Ethernet connection via a built-in Ethernet port and Wi-Fi connection using a MRF24WN0MA[4] module. In Fig 2.3, the layout of the different modules that compose The Things Gateway is shown.

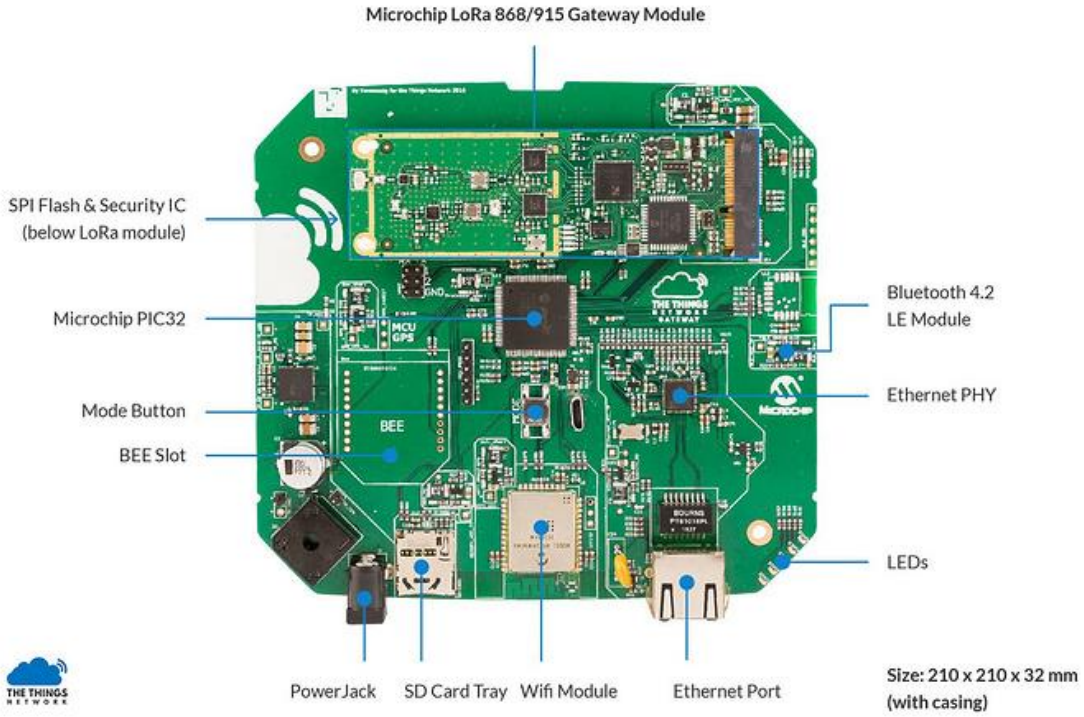

Fig. 2.3 The Things Gateway [18]

2.2. Software

[2] Microcontroller from Microchip
[3] Universal Asynchronous Receiver-Transmitter
[4] Wi-Fi Module from Microchip

In order to develop a LoRaWAN scenario, to use a network server and to program the end-devices, certain software is needed. In this section the software used to develop this project will be detailed.

2.2.1. Network Server

For this project, two free open source Network Servers (NS) have been considered: LoraServer and TTN server. Details of a comparison between the two mentioned NS are shown below.

2.2.1.1. LoRa Server

The LoRa Server [20] is an open-source project, developed by Brocar and sponsored by CableLabs, which provides components for building LoRaWAN networks, including a web-interface and gRPC and REST APIs.

The minimum requirements to run the server are:

- GNU/Linux ARM
- 1GB RAM
- Quad-Core 1.4 GHz
- 8GB HDD

It should be stressed that the requirements above listed are consistent with a Rasberry Pi 3 model B.

LoRa Server provides two API interfaces that can be used to integrate with LoRa App Server, gRPC interface or Restful JSON interface, both interfaces provide exactly the same functionality and authentication mechanism.

2.2.1.2. The Things Network Server

The TTN server [21] is a free server that connects to The Things Network, where the Gateway and end nodes can be managed by console line or via web page.

The minimum requirements to run the server are:

- GNU/Linux ARM
- 1GB RAM
- Quad-Core 1.4 GHz
- 8Gb HDD

It should be stressed that the requirements above listed are consistent with a Rasberry Pi 3 model B.

The Things Network provides an application manager API that offers methods to manage applications and devices registered to TTN. The Application manager

API is exposed by the handler. Two interfaces can be used with the same functionality: either gRPC API or the HTTP API.

2.2.1.3. Comparison Between LoRa Server and TTN Server

In Table 2.1, a comparison between the two mentioned servers is made.

Table 2.1. Server Comparison

Features	LoRa Server	TTN Server
Open Source	Yes	Yes
Connection Between NS-Gateway	Ethernet	Ethernet
Require MQTT	Yes	Yes
Require Redis Database	Yes	Yes
Require PostgreSQL	Yes	No
Gateway Compatibility	Kerlink Laird MatchX Multitech	TTN Gateway Kerlink Laird Cisco Multitech
ADR	Yes	Yes
Type of Node Activation	OTAA & ABP	OTAA & ABP
Compatibility with TTN Gateway	No	Yes
MAC commands supported	Req.status. Adapting data rate of a device. Modify channel settings	Req.status. Adapting data rate of a device. Modify channel settings.
Community	Active	Active

Outcome of the Comparison

As it can be appreciated, the two servers are very similar. The TTN Server has been chosen for this project because it is the only one compatible with our gateway, the TTN gateway. In the following section, a more in-depth explanation of the TTN server is given.

2.2.2. TTN Network Server Architecture

The backend systems of TTN are responsible for routing the IoT data between devices and applications. TTN NS is positioned between the gateway and the applications, and takes care of the routing and processing steps.

In order to decentralize TTN, the network server is split up into several components as shown in Fig. 2.4.

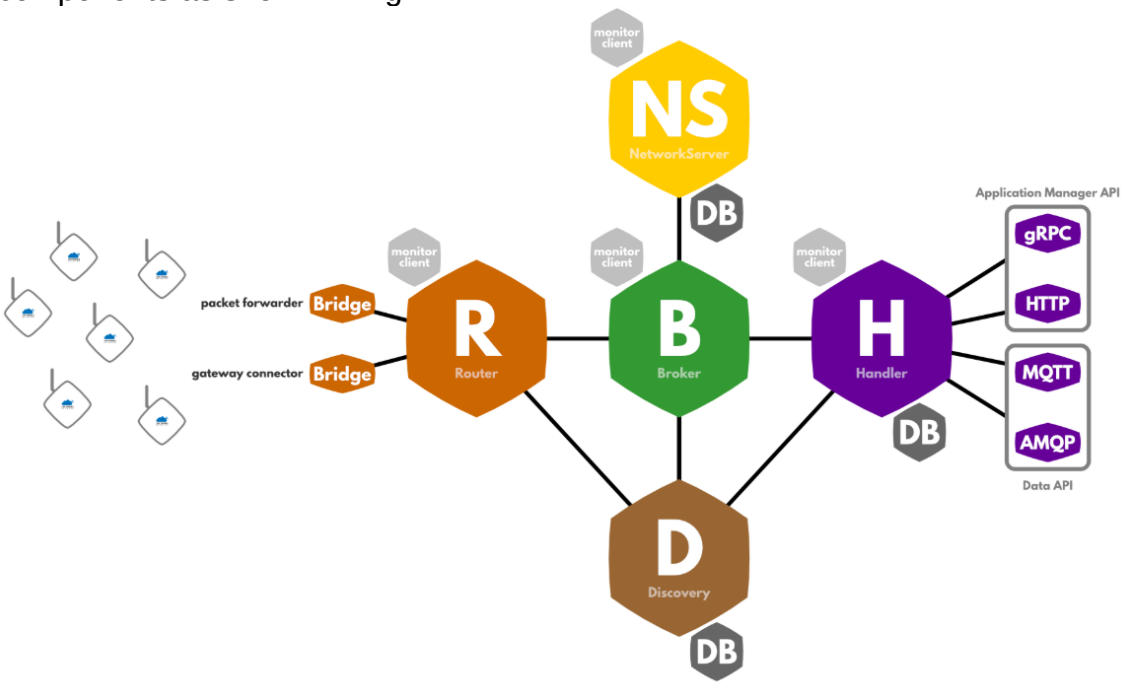

Fig. 2.4 The Things Network Architecture [22]

2.2.3. Components of TTN Network

- **Router:** Is responsible for managing the gateway's status and for scheduling transmissions. Each Router is connected to one or more Brokers.
- **Broker:** Brokers are the central part of TTN. Their responsibility is to map a device to an application, to forward uplink messages to the correct application and to forward downlink messages to the correct Router.
- **Handler:** A Handler is responsible for handling the data of one or more Applications. To do so, it connects to a Broker where it registers applications and devices. The Handler is also the point where data is encrypted or decrypted.
- **Discovery Server:** Helps components to determine where traffic should be routed to. It gives The Things Network Foundation control over which components are allowed to announce specific services.
- **Network Server:** The Network Server is responsible for functionality that is specific for LoRaWAN.

The way the whole system works is as follows: Nodes broadcast LoRaWAN messages over the LoRa radio protocol. These messages are received by the gateway. The gateway is connected to the Router. The router is connected to the Broker. The Broker connects to the Handler and the Handler to the Application server by using Message Queuing Telemetry Transport (MQTT) API which is explained next in 2.2.3.1 [22].

2.2.3.1. MQTT

MQTT Is a machine-to-machine, widely used for IoT communications, connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport.
The Things Network uses MQTT to publish device activations and messages, but also allows publishing a message for a specific device in response [23].

2.2.3.2. Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. JavaScript is used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser [24].

2.2.4. Arduino environment

As mentioned before, the end-device in this project is a Dragino shield running on an Arduino Uno. All the programming and configuration of the end-device is done in the Arduino IDE using a library developed by IBM called LMIC [25].

LMIC

LMIC stands for LoraMAC-in-C. It provides LoRaWAN class A and class B support. It works with the American and European frequency band. Some of its main features are:

- Sending packets uplink, taking into account duty cycling.
- Encryption and message integrity checking.
- Receiving downlink packets in the RX2 window.
- Custom frequencies and data rate settings.
- Activation by Personalization (ABP)
- Over-the-air activation (OTAA / joining).

This library is intended to be used with plain LoRa transceivers, connecting to them using SPI. In particular, the SX1272 and SX1276 families are supported.

CHAPTER 3. LoRaWAN Network Deployment

In this chapter, the development of a LoRaWAN for an educational scenario with the use of a TTN gateway is detailed. In the first section, a guide on how to activate The Things gateway is shown, followed by the process on the activation of an end-device. It should be stressed that before working in the deployment of a LoRaWAN a TTN account is needed.[5]

3.1 Development of a private Back-End

There are two ways to manage the LoRaWAN network with TTN gateway:

- **A generic console** created by The Things Network where their own router, handler, broker and network server is implemented.
- **A private Backend** where the management of the gateway is done via the command line interface and where the user or owner of the gateway implements the router, handler, broker and network server.

3.1.1 Generic TTN Console

The Things network offers a graphical user's interface (TTN Console) where applications, devices and gateways can be managed and monitored. The selection screen can be seen in Fig 3.1

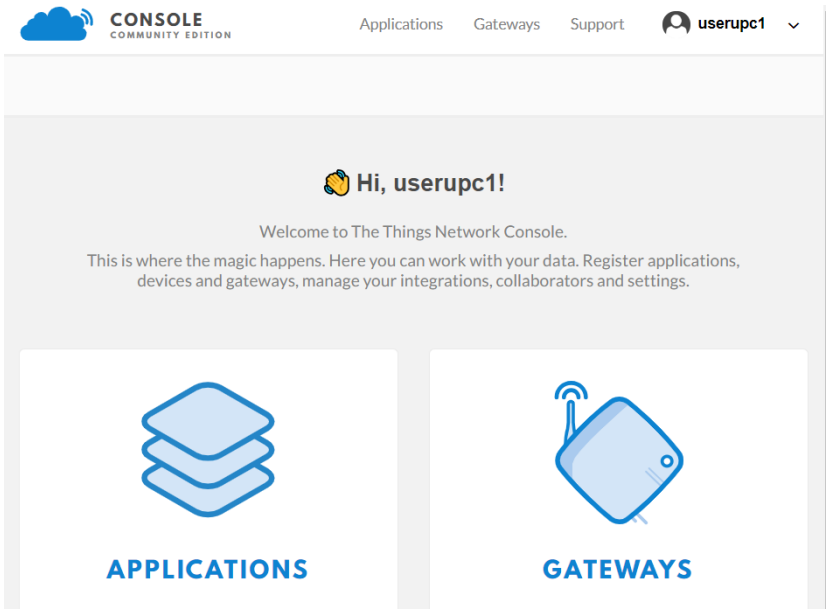

Fig. 3.1 The Things Network Console Home screen

There is the possibility to choose between applications and gateways.

[5] TTN Account registration link: https://account.thethingsnetwork.org/register.

3.1.1.1 Applications

When the application option is selected, it will display all the applications that have been created by the user. Under each application different end-devices can be registered, via OTAA or ABP. In addition, the traffic that the application is receiving can also be monitored. In Fig 3.2, the application layout with their different features can be seen.

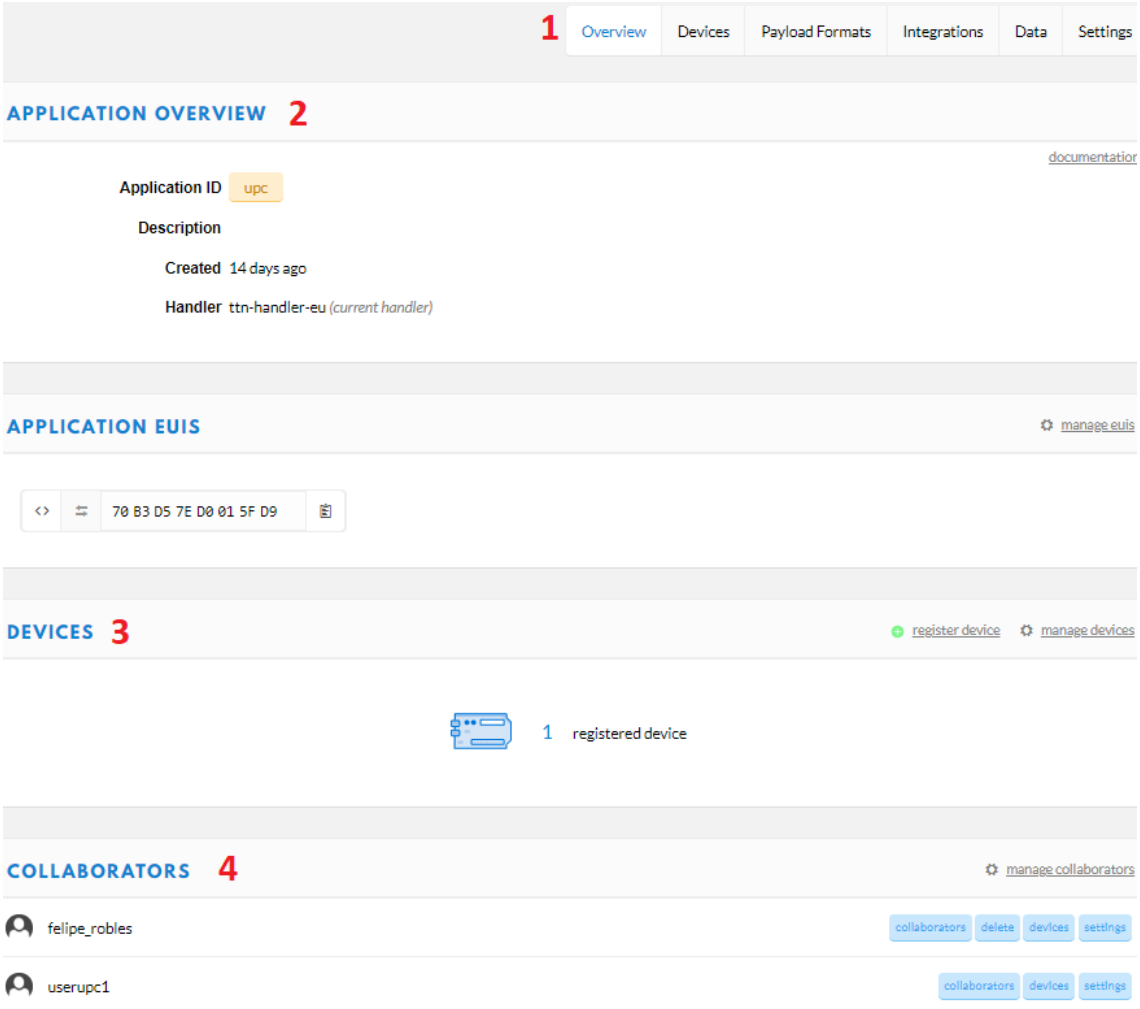

Fig. 3.2 TTN Application Layout.

1) **Application Features**. In this section, the different features that the application offers are shown.
   - **Devices:** It shows all the different Devices that have been registered to the application.
   - **Payload Formats:** It offers a quick way to configure and convert the different payload the applications receive.
   - **Integrations:** This section connects the network to different APIs that support TTN.
   - **Data:** all the data (uplink and downlink) from/to the devices associated with the application can be monitored.
   - **Settings** this tab is where all the parameters can be tuned.

2) **Application Overview.** In this section, a summary of the application is shown.
3) **Devices**. Here, an end-device can be registered or deleted.
4) **Collaborators.** All the users that have access to the application and their privileges.

3.1.2 Private Back-End

Apart from the TTN Console there is also a way to access the same features but having a private router, handler, broker and network server. All the features have to be managed and seen only in command line interface. In order to set-up the private environment there are some software tools that have to be installed and running. The working environment is running in a native Ubuntu 18.04 environment. The requirements are:

- Have Redis installed and running.
- Have Mosquitto installed and running.
- Have ttn (masterbranch) downloaded.
- Have Node.js installed
- Have gateway-connector-bridge (master branch) downloaded.

It should be stressed that all the previous requirements can be found online [26].

All the steps and scripts for the creation of a private back-end are detailed in annex 1.

After everything is started the network can be managed with the line command “ttnctl”.

All the features that are available for the graphic environment that was mentioned before is also available in the private environment.

It should be stressed that when setting up the environment, it’s not private as a whole because it still depends on TTN account servers and router to have access to the complete LoRaWAN network. The user must have a TTN account and access the account using the command line interface (CLI).

As mentioned before, the gateway that is used is the TTN gateway which in the current version is incompatible with the private backend. Tests have been done in order to try to have a fully private network server using the TTN gateway with different approaches:

- **Creation of a Domain Name System (DNS):** As the gateway connects to a TTN router using a URL which is resolved by a DNS server and connects it to the router’s IP. Having this information, a DNS server was created in effort to forward the gateway to connect to the private router that was created. This couldn’t be achieved due to that TTN servers and the TTN gateway exchange digital certificates.

- **Sniffing traffic:** Another test was sniffing the traffic the gateways send to Internet using the PC as an Access Point for the gateway. The packets were captured but they were encrypted and not much couldn't be done with this information.

The only current solution in order to have a fully private environment is to use another gateway with Semtech's packet forwarder compatible with TTN. In upcoming stack versions (currently V2) TTN is planning to release a new firmware for the TTN gateway and NS which will provide a fully private network.

3.2 The Things Network Gateway Activation

As mentioned before, the gateway is the central part of the Network so the first step for the development of the network was to activate the gateway on TTN.

3.2.1 Steps for Gateway Activation

The gateway needs to connect to the Internet in order to achieve the full installation, the gateway has two ways of connecting to Internet: via Wi-Fi or Ethernet. The steps for activating the device are:

1. Go to activate.thethingsnetwork.org
2. Register the gateway, giving it a ID and selecting a frequency plan that the gateway will use depending on the Location. Figure 3.3

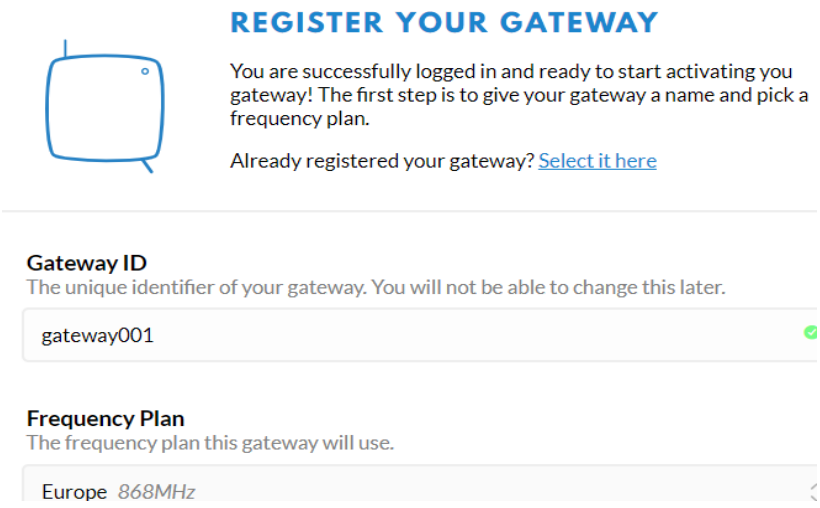

Fig. 3.3 Gateway Activation Step 2.

3. Connects the PC that is doing the configuration to the gateway. This process is over Wi-Fi where the gateway acts as an access point. Figure 3.4

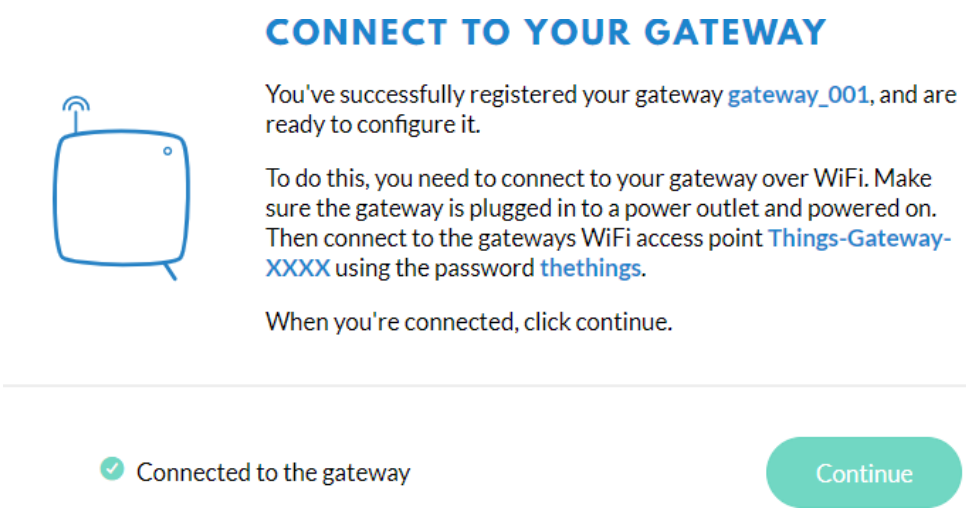

Fig. 3.4 Gateway Activation Step 3.

4. In the next step the configuration of the gateway is made, at this point the type of connection to the network server is established, it can be either Wi-Fi or Ethernet. If Wi-Fi is selected, it is necessary to select the Wi-Fi ID and its password. If Ethernet is selected, nothing is needed more than connecting the gateway to an access point via Ethernet cable. Figure 3.5

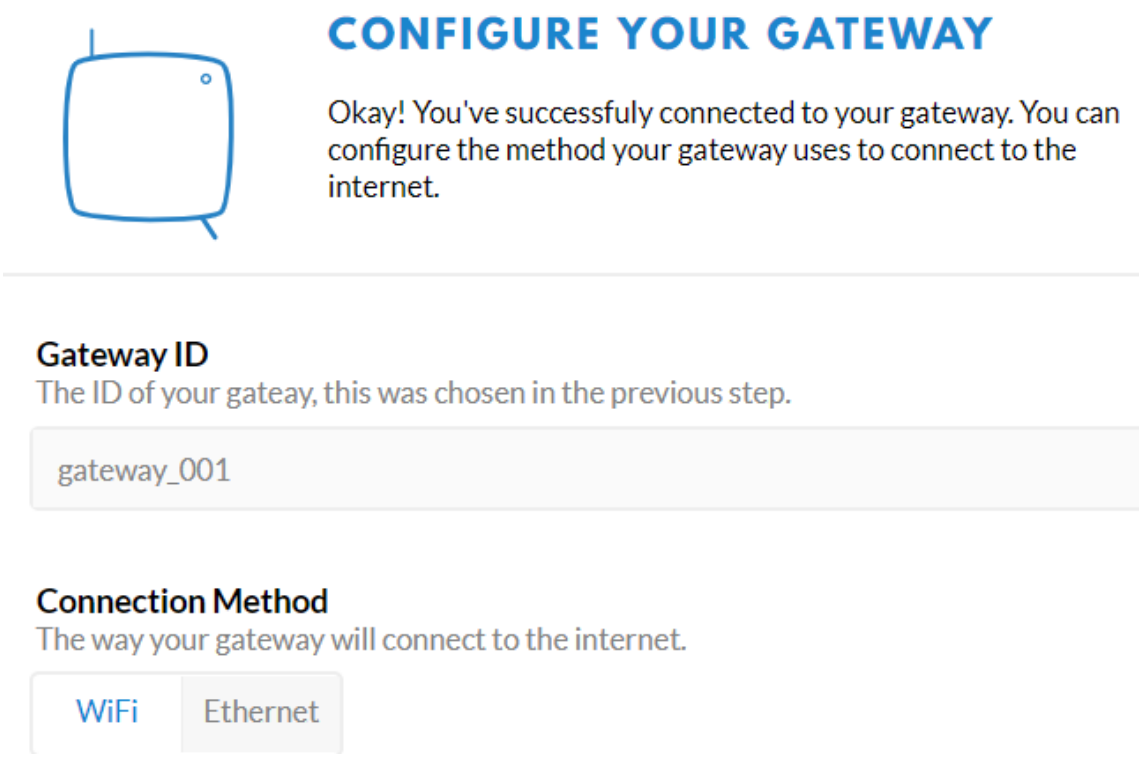

Fig. 3.5 Gateway Activation Step 4.

5. Check that the connection succeeded. Figure 3.6

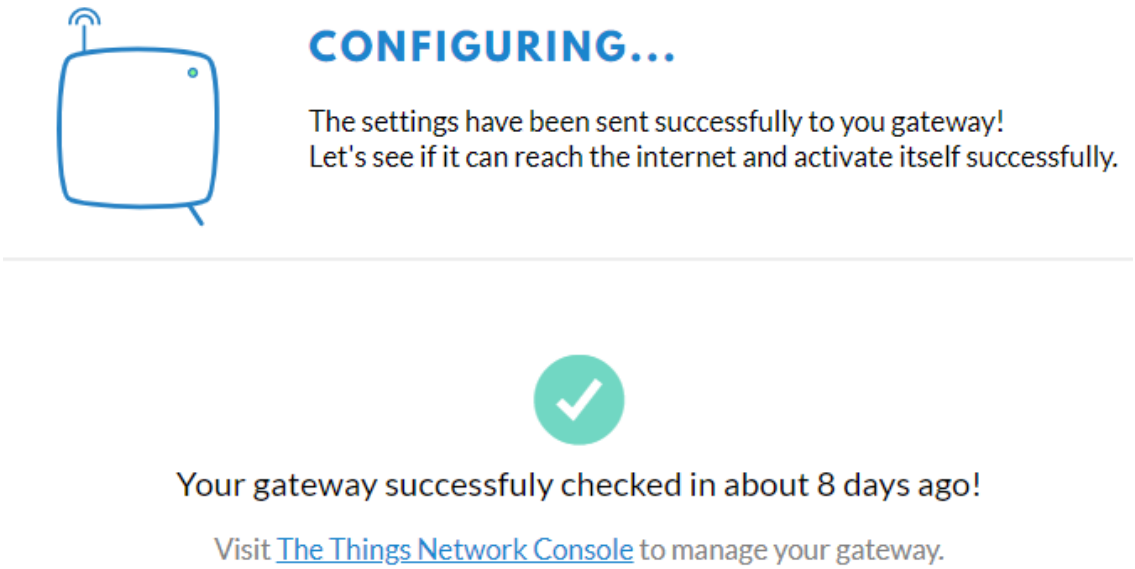

Fig. 3.6 Gateway Activation Step 5.

Some errors may arise in the activation process and in order to have an idea of what may the problem be the gateway counts with status Led indicators. In table 3.1 the meaning according to the status led code, depending on the state of the LED and the blinking, can be observed.

Table 3.1. Gateway Led Code

LED 1	LED 2	LED 3	LED 4	STATUS
On	Slow Blink			Connecting to the internet
On	Fast Blink			Could not connect to the internet
On	On	Slow Blink		Activating
On	On	Fast Blink		Could not activate (restart activation from step 1)
On	On	On		Activated
On	On	On	On	Connected to the server

3.2.2 Gateway Configuration on Network

Once the gateway is activated the different parameters have to be configured, in order to configure the parameters, go to settings tab (see Figure 3.7, left side).

The parameters to be configured are:

Location: The latitude and longitude have to be set.
Privacy: Make the status, location and the owner public or private.

Router: Select the router that the gateway will connect to.
Auto-update: An option to automatically update the gateway with the latest firmware rom TTN.
Collaborators: A section where the ownership of the gateway and inclusion of users that can access the gateway.

An owner can add collaborators for the gateways and can manage its privileges. The rights that can edited for each collaborator are:

- **Gateway settings:** Manage the gateway settings and access keys.
- **Gateway collaborators:** Edit the gateway collaborators
- **Gateway delete:** Delete the gateway
- **Gateway location:** View the exact location of the gateway
- **Gateway messages:** View the traffic that occurs on the gateway

The owner can select these parameters, by marking them in a check box, when going through the Collaborators' option in the gateway settings configuration. This option can be seen also in Figure 3.7.

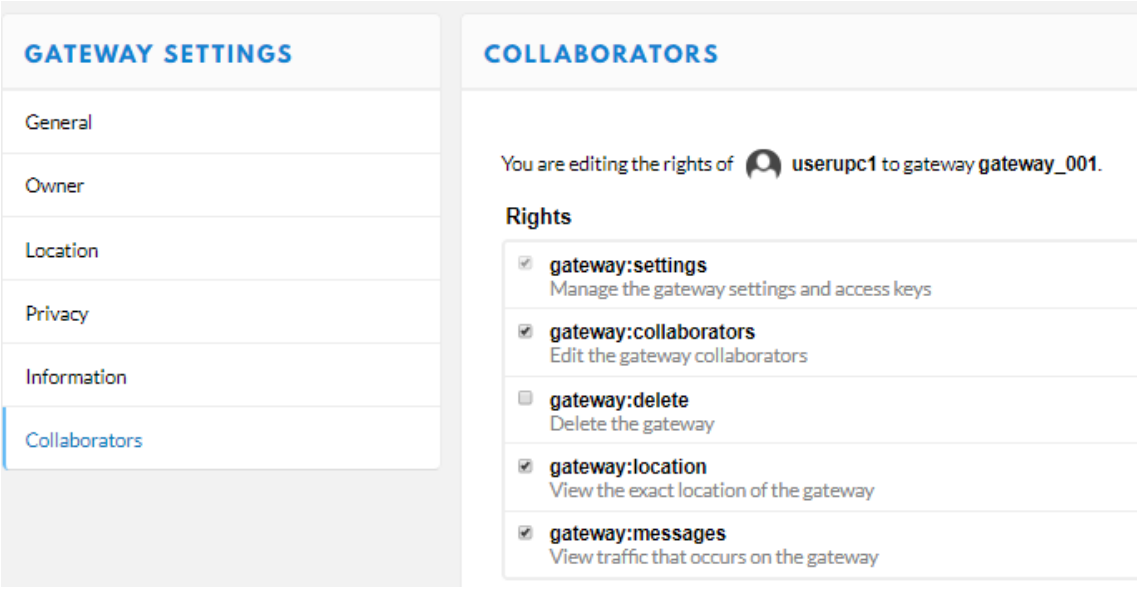

Fig. 3.7 Gateway Settings Configuration.

3.3 End-Device Activation

After activating and configuring the gateway it is necessary to activate and configure the end-device. In order to activate the node first is necessary to create an application on the TTN console and then proceed to register and activate a device.

3.3.1 Application Creation

Login in to a TTN account that is associated with the gateway, go to the applications Tab, select add application, assign an ID and select the handler where the application is going to be registered to, in this project case it is registered to *ttn-handler-eu.* Finally click on add application button.

When the application is created, a unique identifier (EUI) is assigned to the application.

In the Application it is also possible to have collaborators, a user being a collaborator on the gateway doesn´t imply that it will have access to the application, as being an application collaborator doesn´t imply that it will have access to the gateway. To assign collaborators to the application go to settings tab in the application layout page and select collaborators, add a collaborator by username and check the parameters the collaborator should have access to. Figure 3.8 shows the parameters that can be configured.

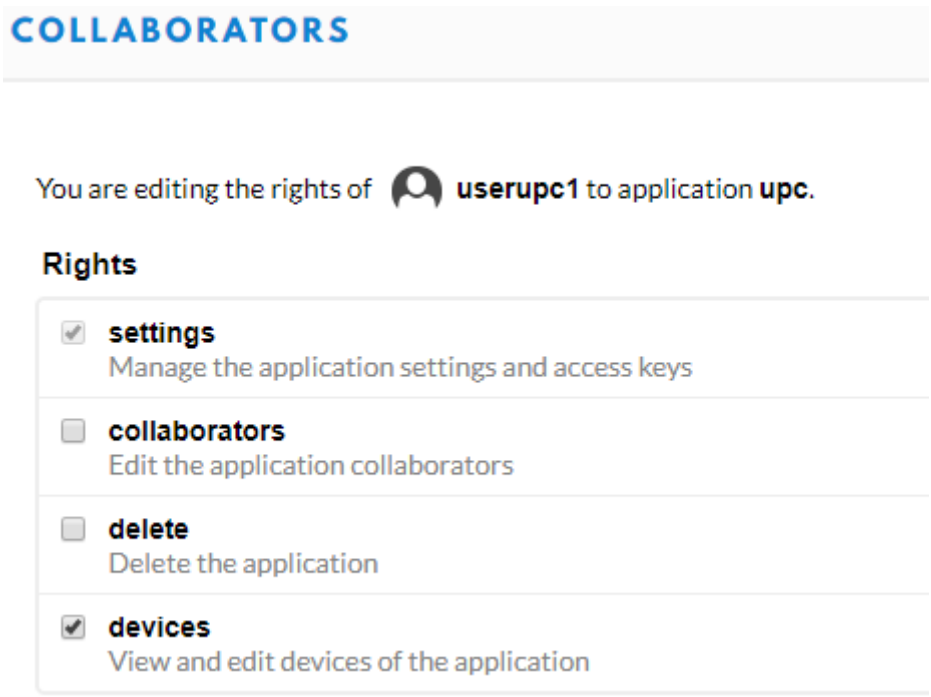

Fig. 3.8 Collaborators permissions.

3.3.2 Device Registration

Having the application created, the next step is to register the device to the application.

In the application layout page, in the device section register, select register device and proceed to fill in the Device ID field. Device EUI and App Key are generated by TTN.

3.3.3 Device Activation

After a device is registered, the final step in order to send and receive packets is to activate the node. In order to have a better understanding of the actions needed to activate the device, the layout of the device console page is shown in Fig.3.9.

To activate the device, the node has to be programmed. As explained in section 2.2.4, this is done using the Arduino IDE. The whole code can be seen in annex 2. The important parts of the code will be explained in this chapter.

The activation type selected for this case is ABP. The Network session key and the App Session key that is under DEVICE OVERVIEW in the TTN console are used to activate the device. These parameters have to be declared in the Arduino IDE. A way to declare them is:

```
static const PROGMEM u1_t NWKSKEY[16] = { 0x51, 0x9F, 0x76, 0x59,
0x40, 0xD2, 0xC0, 0x54, 0x16, 0x25, 0x9F, 0x47, 0x30, 0x41, 0x8C,
0x8A}

static const u1_t PROGMEM APPSKEY[16] = { 0x62, 0x55, 0x88, 0x34,
0xD1, 0x78, 0x87, 0xFA, 0x1E, 0xB2, 0xE3, 0xE1, 0x1A, 0x70, 0x1E,
0xFE}
```

It should be stressed that OTAA device activation hasn't been tested in the project. Future lines of work can try this kind of activation.

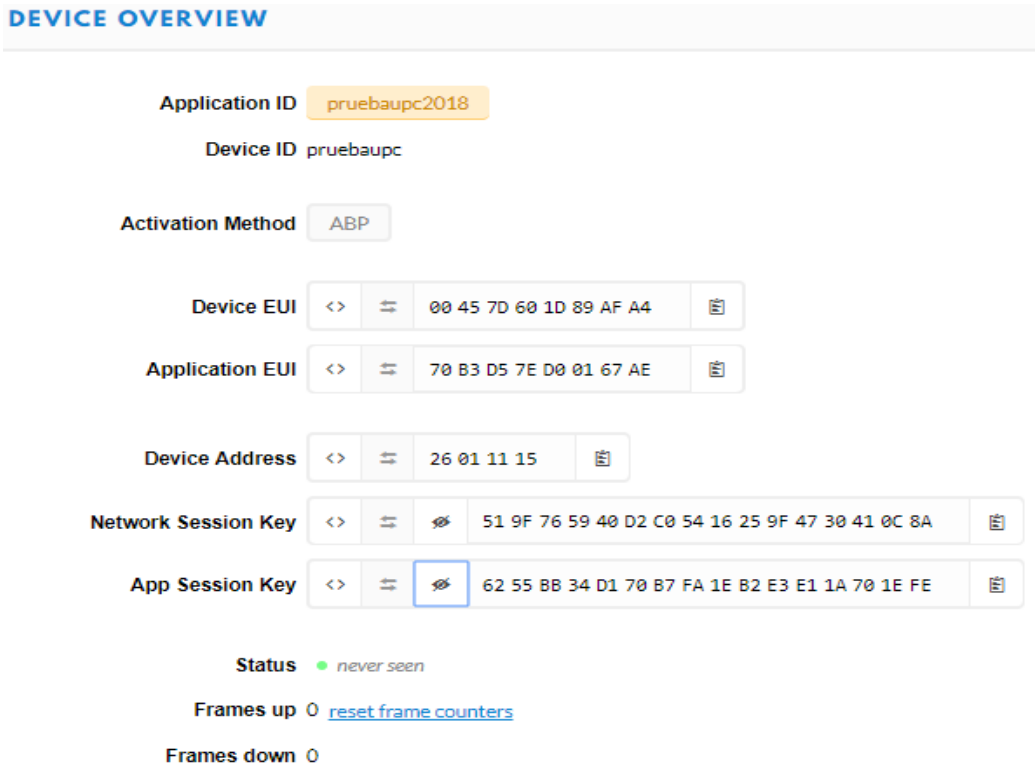

Fig. 3.9 TTN Device Layout.

The device address is generated by the TTN console and also has to be declared in the ARDUINO IDE.

```
static const u4_t DEVADDR = 0x260117DC;
```

If all the parameters are correct, the device will correctly join the network.

CHAPTER 4. Test Environment

This chapter contains the guides and explanation of the possibilities that can be exploited in an educational environment in order to understand the different parts of a LoRaWAN environment such as: LoRaWAN key concepts, payload formats, configuration of end-devices, different data rates and transmission power, among others parameters.

4.1 Payload Analysis

Once the device and the gateway are up and running, the next step is to send data from the device to the Network Server. The device will send continuous packets to the gateway with different data rates, depending on the spreading factor that is selected. The gateway will capture these packets and forward them to the corresponding network server.

To transmit a message, it is necessary first to configure the node on the Arduino IDE. First it is necessary to tell the node what message is going to be transmitted. In this case the message to be transmitted for testing purpose is “22222”. The code for setting up the payload on the ARDUINO IDE is:

With the following command, it declares the payload as an unsigned 8-bit value.

```
static uint8_t mydata[] = "22222";
```

With the following line, it sets a new transmission packet using the LMIC library tool setTxData2, the first and the last parameter represent the port and whether the message requires confirmation (1) or not (0).

```
LMIC_setTxData2(1, mydata, sizeof(mydata), 0);
```

The complete code can be seen on annex 2. It is also needed to specify the spreading factor and transmit power. The parameters chosen in this case are SF=7 and TX=14 dBm. To specify these working factors, it is necessary to use the tool setDRTxpow from the LMIC library, as it can be seen in the following line of code:

```
LMIC_setDrTxpow(DR_SF7,14);
```

4.1.1 Gateway Payload Analysis

To analyse the payload that the gateway is receiving, go to the gateway section and select the payload tab. In Figure 4.1, it can be seen all the packets that the gateway is receiving with information regarding the sender, the size of the packet, the airtime, the data rate and the modulation that is using.

It should be stressed that the gateway captures all the packet from nearby nodes even if the node is not part of the network we are working on. At this point this

packets are encrypted and the gateway will forward them to the network server or contrary, they will be dropped if there is no information related to them.

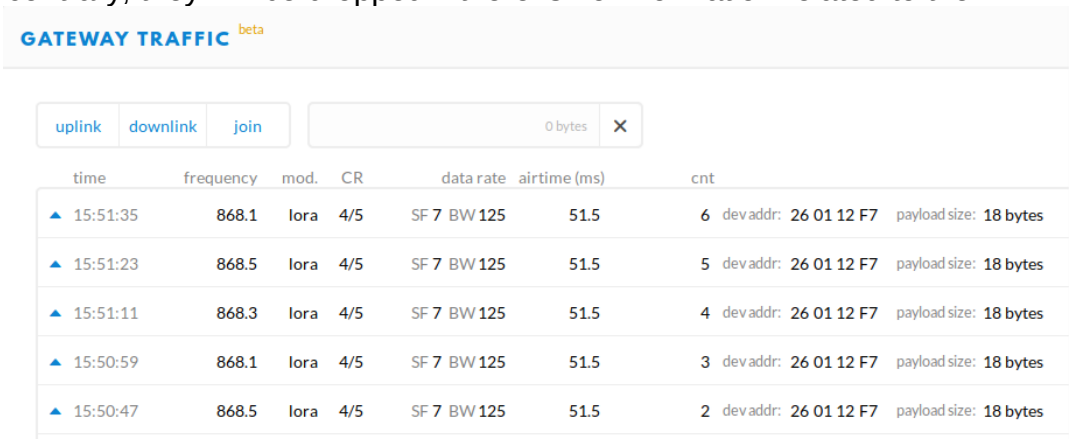

Fig. 4.1 Gateway Traffic.

Furthermore, the physical payload of the packet and the device that sent it can also be accessed by clicking on any incoming packet from the gateway traffic section (see Figure 4.2).

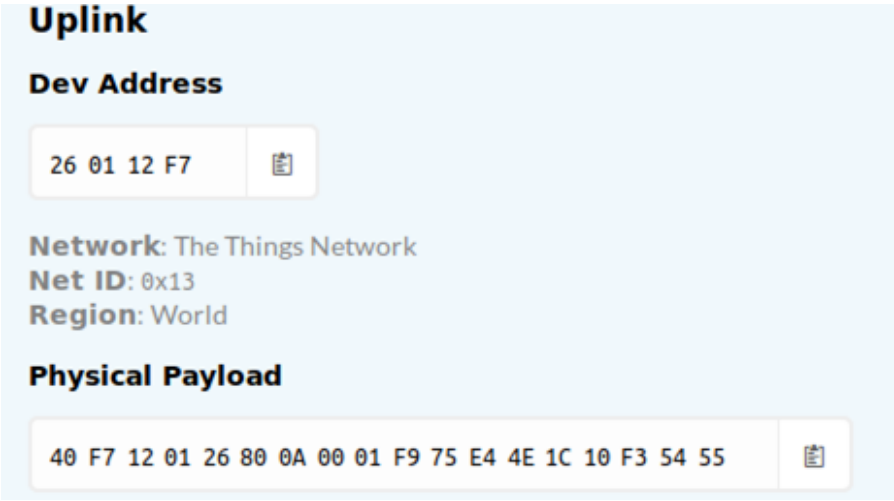

Fig. 4.2 LoRa Physical payload sent from node.

Continuing with the information that can be accessed by selecting any packet from the gateway traffic, it is possible to have information about the link such as: SF, BW, airtime, coding rate, and the frequency. (Seen in Figure 4.3)

It can be appreciated that in line 3, in figure 4.3, under payload the value is not the “22222” that was sent. This is because until this point the payload is encrypted.

Two other important parameters that this view gives are the as the Signal-to-noise ratio (SNR) or Received Signal Strength Indicator (RSSI) which are link quality indicators, which can be used for future lines of work regarding link coverage and interference affection on the link.

Apart from obtaining all the information for the different parameters previously mentioned, the console also shows the trace of the packet, the steps the packet took to arrive to the gateway and the different times it took to complete this actions.

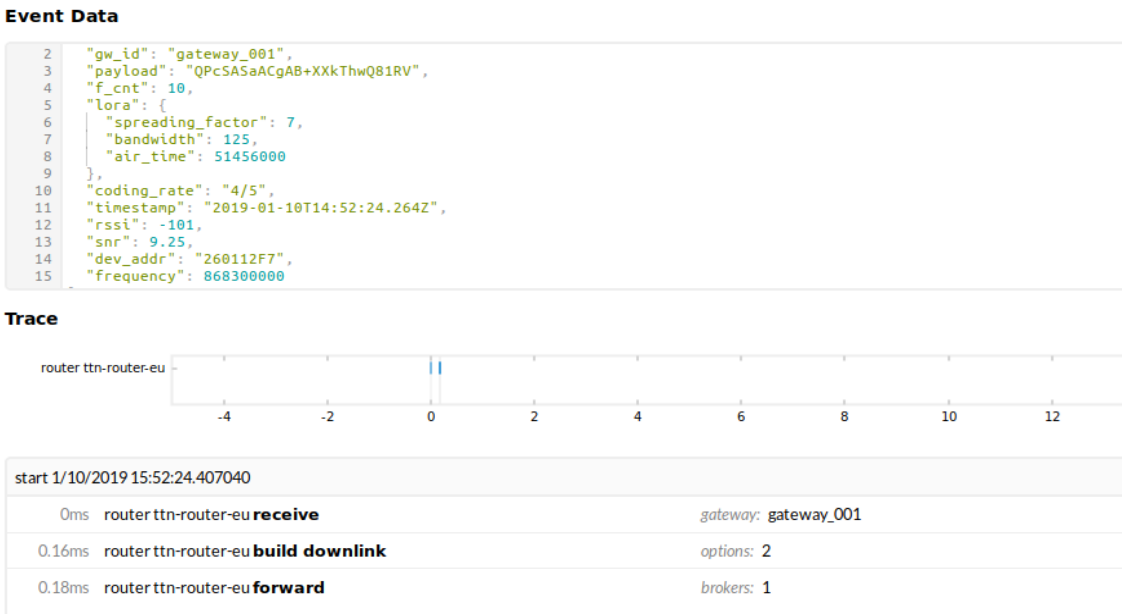

Fig. 4.3 LoRa Physical payload sent from node.

As mentioned before, the TTN console only shows the physical payload and the different parameters. Using Node.js it is possible also to analyse in a more in-depth way the LoRa packet and its structure. To do this, it is necessary to obtain the physical payload from the gateway traffic and generate a small Node.js script in which it is necessary to indicate the nwSKey, appSKey and the physical payload and run it in the CLI using the “node” tool. In Figure 4.4 it can be seen the outcome of running the script. The full script can be seen in annex 3.

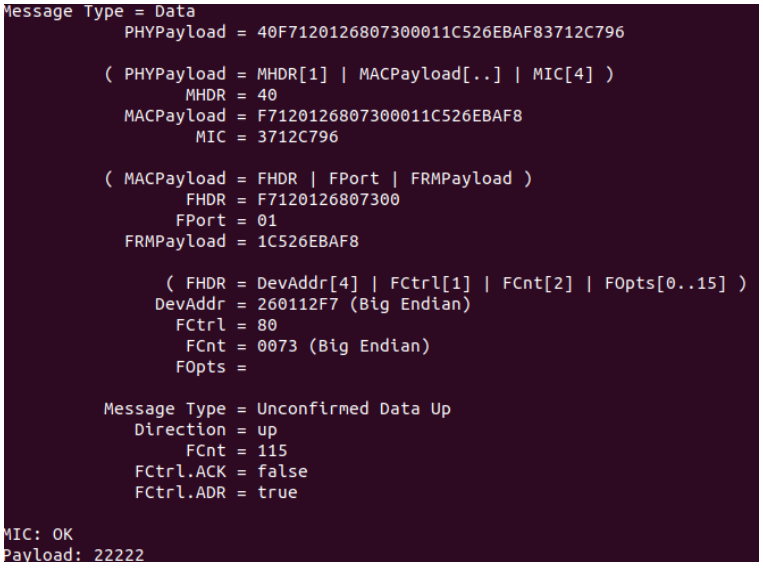

Fig. 4.4 LoRa Physical payload decoded in Node.js.

In Figure 4.4 the breakdown of the LoRa physical payload, its components and their value are shown. It also shows the attributes that the packet has (i.e. MIC, ADR, ACK). In the last line the message sent from the end-device can be read in plain text (“22222”).

4.1.2 Applications-Device Payload Analysis

Apart from analysing the payload that the node sends to the gateway, it is also possible to analyse the traffic that the gateway sends to the network server which is the traffic that is fully unencrypted. To access the device data first go to application section on the TTN console then click on the registered devices section and finally on the device wanted. Figure 4.5 shows the layout page for the device’s data traffic.

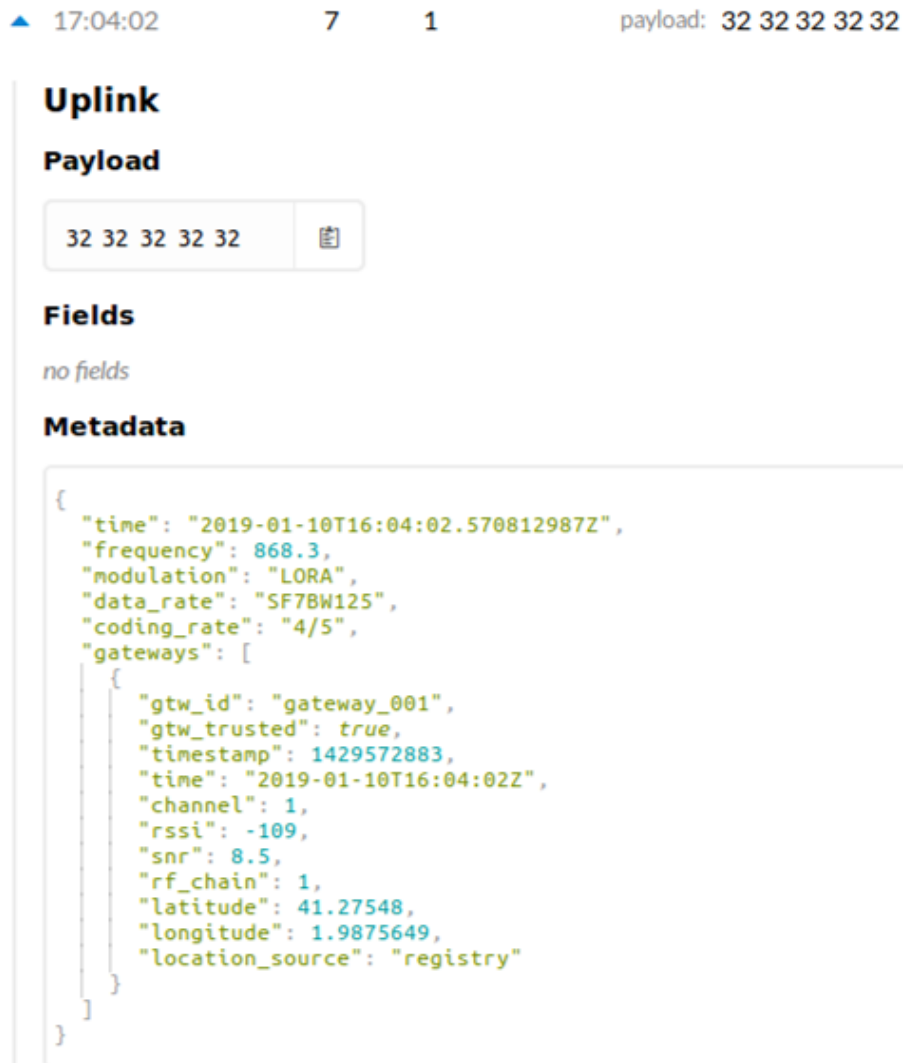

Fig. 4.5 Device traffic.

As it can be seen in Figure 4.5, the payload is not encrypted in the device traffic analysis although it is in hex format “3232323232” which in string value is

"22222", also the data that comes in from the gateway with the payload can be analyzed. Besides the fields that could be observed in the gateway traffic analysis, in section 4.1.1, there are more fields added that are mainly information of the gateway status and the link quality as the SNR or RSSI.

4.1.3 Sending Downlink data

Messages to the end node can also be sent from the TTN console. In order to send data to the node, first go to the application section on the TTN console. Then click on the registered devices section and finally on the device wanted. After this, scroll down to the Downlink section (Fig. 4.6).

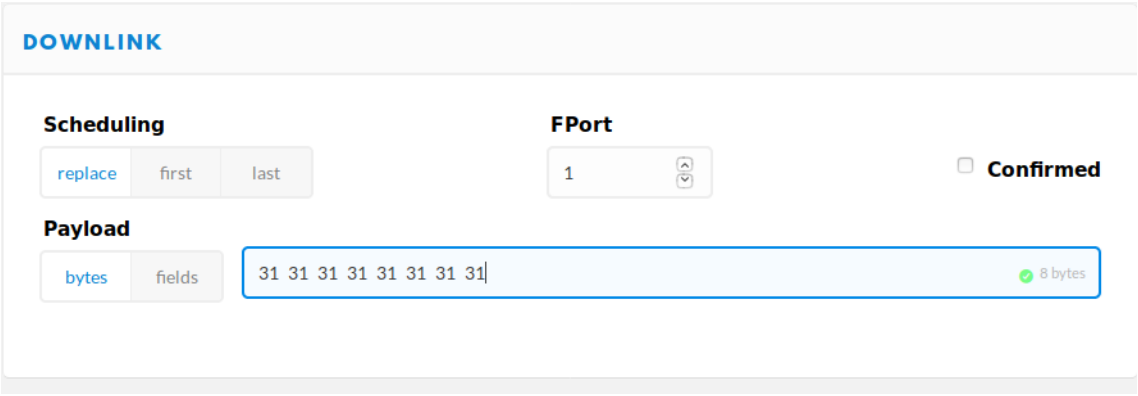

Fig. 4.6 Downlink Generator

The downlink data packet can also be analyzed the same way the uplink packet was.

Downlink packet Gateway
To access the traffic data, go to the gateway section then to the data tab and select the downlink option. In Figure 4.7, the outcome that should be obtained is shown.

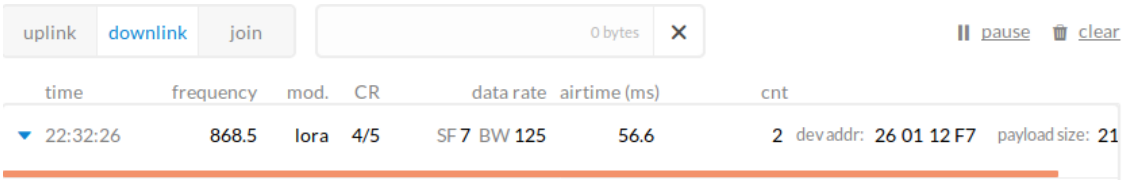

Fig. 4.7 Downlink message in Gateway

As uplink messages in the gateway, the downlink payload is also encrypted as can be seen in Figure 4.8, line 3, under the payload field. Apart from the payload, it is also possible to see some parameters that the network server has established and the device address where the LoRa packet has to go.

Physical Payload

```
A0 F7 12 01 26 00 02 00 01 3F 49 90 7D 7A E4 3A 92 BB A4 50 02
```

Event Data


```
{
  "gw_id": "gateway_001",
  "payload": "oPcSASYAAgABP0mQfXrkOpK7pFAC",
  "f_cnt": 2,
  "lora": {
    "spreading_factor": 7,
    "bandwidth": 125,
    "air_time": 56576000
  },
  "coding_rate": "4/5",
  "timestamp": "2019-01-14T21:32:26.467Z",
  "dev_addr": "260112F7",
  "frequency": 868500000
}
```


Fig. 4.8 Downlink message in Gateway

As it was the case in the uplink message, also the physical payload can be analyzed with Node.js and the result is shown on Figure 4.9

```
Message Type = Data
            PHYPayload = A0F7120126000200013F49907D7AE43A92BBA45002

          ( PHYPayload = MHDR[1] | MACPayload[..] | MIC[4] )
                 MHDR = A0
            MACPayload = F7120126000200013F49907D7AE43A92
                   MIC = BBA45002

          ( MACPayload = FHDR | FPort | FRMPayload )
                 FHDR = F7120126000200
                FPort = 01
            FRMPayload = 3F49907D7AE43A92

              ( FHDR = DevAddr[4] | FCtrl[1] | FCnt[2] | FOpts[0..15] )
             DevAddr = 260112F7 (Big Endian)
               FCtrl = 00
                FCnt = 0002 (Big Endian)
               FOpts =

          Message Type = Confirmed Data Down
             Direction = down
                FCnt = 2
            FCtrl.ACK = false
            FCtrl.ADR = false

MIC: OK
Payload: 11111111
```

Fig. 4.9 Downlink message with Node.js

4.1.4 Trace & Time Analysis

The time and the trace for uplink and downlink messages can also be analyzed. In this section, the analysis of these parameters for an uplink message requiring an ACK is made.

In order to access the trace of the packet, it is necessary to go to the gateway traffic section and select the 2 last downlink messages and head to the trace section (Figure 4.10).

For testing and analyzing purposes the end-device sends a message that requires an ACK from the NS.

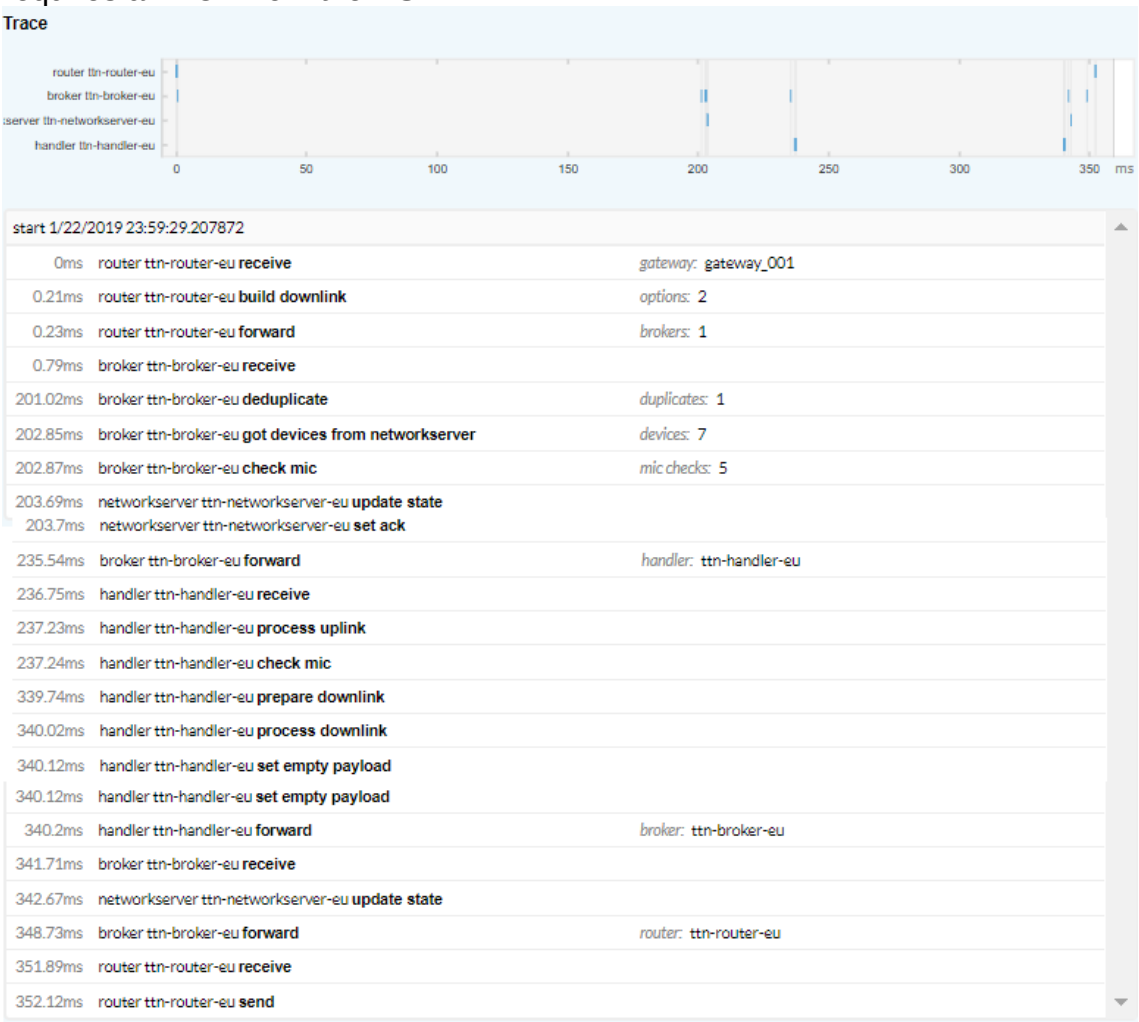

Fig. 4.10 Trace and Time Analysis

Only the downlink trace is necessary because the uplink time and trace is also taken into account in the downlink trace. The uplink time and trace starts once the packet arrives at the NS. It's not possible to measure the time it takes for the packet to go from the end-device to the network server with this procedure. It only measures the time the NS takes to receive, process and send the messages to the gateway.

In Figure 4.10 it can be observed of how the packet traveled in order to generate a downlink message, passing through all of the network server components. Furthermore, the total processing time can be obtained in this case (it is 352.12 ms).

Once an uplink packet arrives to the gateway, it will be forwarded to a router which is in charge of receiving the LoRa packet from the gateway, building the downlink packet and forwarding it to the corresponding Broker according to the region of work.

The Broker will receive the packet and make a deduplication process, which is a process whereby the broker has to check if that packet is delivered only once to the NS. The broker will determine the exact device that sent the message, and the application it belongs to. To do this, the backend has to perform a series of MIC checks, one for each device that uses the same device address. To do so, the Broker requests a list of devices with the given device address from the NS and checks if the MIC can be validated using network session key.

The broker sends the message to the NS so that the device´s state can be updated. The NS also adds a downlink template to the message. This template is later used by the handler to send the downlink message to the end-device. It contains all necessary values (such as the frame counter, message type and option flags) so that the Handler only has to add the application payload to the message. In this case, it sets the ACK.

The NS forwards the packet to the Broker which in turn forwards the message to the handler. The handler receives, decrypts and publishes the uplink messages on an MQTT and checks the MIC. After publishing the uplink message to MQTT, the Handler will determine whether it is necessary to reply to the device with a downlink message. In this case a confirmation is required so the handler is in charge for preparing and building the downlink message. As it is an ACK, the payload is set as empty.

The Handler sends the downlink message back to the Broker, after the Broker receives the downlink message, it sends the message to the NS, which will update the device´s state (specifically, the frame counters) and generates the MIC of the message. After this, the Broker forwards the downlink message to the router that is responsible for sending the message to the gateway.

In Figure 4.11 the complete process of an uplink and downlink trace can be observed. It shows all the elements, their respective functions and the delay time for each component.

In Figure 4.11 the network components are represented as:

- G: Gateway
- R: Router
- B: Broker
- NS: Network Server
- H: Handler

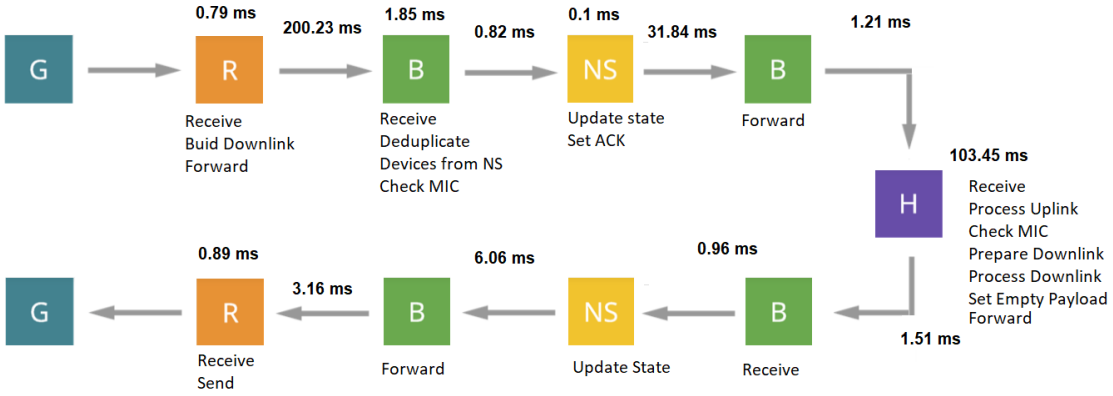

Fig. 4.11 Delay and Component Analysis

Until now, only the network processing time is analyzed, in where the time taken into account is from the gateway transmitting uplink messages until the gateway receiving downlink messages. The Round-trip time (RTT) is the total time since a packet is sent from the device, until the ACK signal is received. In order to obtain the RTT, an analysis of the airtime has to be made. To do so, the Arduino serial monitor along with programming functions are used.
The Arduino function used to calculate the RTT is “milis()” which records the time at which an action took place. With the help of this function two variables were created:

- startTime: Variable where the time when the packet is sent is stored.
- endTime: Variable where the time when the ACK signal is received is stored.

In this way, the RTT can be calculated as the difference between endTime and startTime.

The average RTT obtained is 1106 ms, which represents the network processing time plus the air transmitting time. In Figure 4.12, the outcome in the Arduino serial monitor is shown.

```
Sending Packet
Start time:    141777
EV_TXCOMPLETE (includes waiting for RX windows)
End time:    142884
RTT (milliseconds):    1107
Data Received:
Sending Packet
Start time:    152944
EV_TXCOMPLETE (includes waiting for RX windows)
End time:    154050
RTT (milliseconds):    1106
```

Fig. 4.12 Arduino Serial Monitor Time Analysis

4.1.5 Spreading Factor analysis

Finally, the working principle of SF, and its impact on the RTT, can also be analysed with the use of the RTT tool developed in section 4.1.4. In order to make the tests 50 uplink messages with ACK have been generated for each of the following scenarios:

- SF = 7 Payload = 24 bytes.
- SF = 7 Payload = 51 bytes
- SF = 9 Payload = 51 bytes
- SF = 12 Payload = 51 bytes

For testing and analyzing purposes the ADR in the uplink is set as off, otherwise TTN would automatically adapt the SF to the fastest DR possible according to the link parameters. In order to set the ADR of the following line has to be implemented in the Arduino code for the end-device:

```
LMIC_setAdrMode(0);
```

The gateway and the end-device are placed in a distance of 5 meters between each other with a direct line-of sight and both with their respective antennas and a TX = 14 dBm. All the time values in the following figures and in annex 4 are in ms.

SF = 7 Payload = 24 bytes

The SF has to be set on the end-device through the Arduino IDE with the use of the setDRTxpow tool explained in section 4.1.

From the 50 samples taken (annex 4) it can be observed that the RTT for the majority of them (45) is similar, having values between 1105 ms – 1108 ms. However, 5 values have a higher RTT (i.e. the max RTT = 7275 ms). This values were presented because the end-device didn't receive an ACK so it keeps resending the message until it obtains a response from the NS. In Figure 4.13, the TTN console with a report for two packets (one with a normal functioning and another with a retry), is shown:

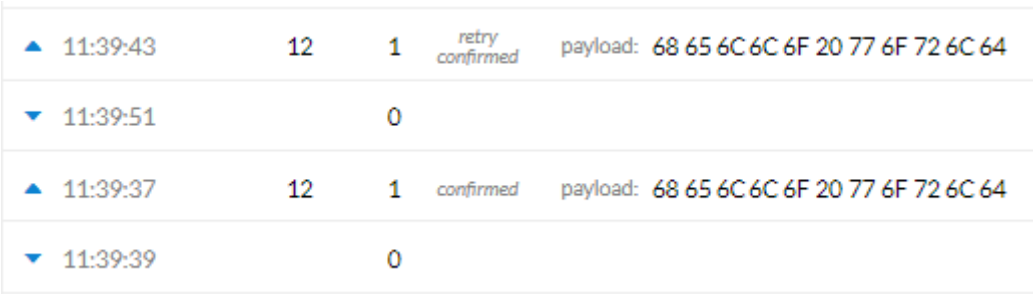

Fig. 4.13 Packet confirmation

For this particular case with the RTT analysis the two packets, shown in Figure 4.13, obtained a RTT = 7273 ms for the packet that was resent and RTT=1108 ms for the packet that functioned normally. These times can be contrasted with the TTN clock which shows that for one packet the uplink and downlink process took from 11:39:43 until 11:39:51, having an estimate of 8 seconds. For the other packet, it took from 11:39:37 to 11:39:39 having an estimate of 2 seconds.
In Figure 4.14, a graphical representation of the 50 samples as a summary is shown.

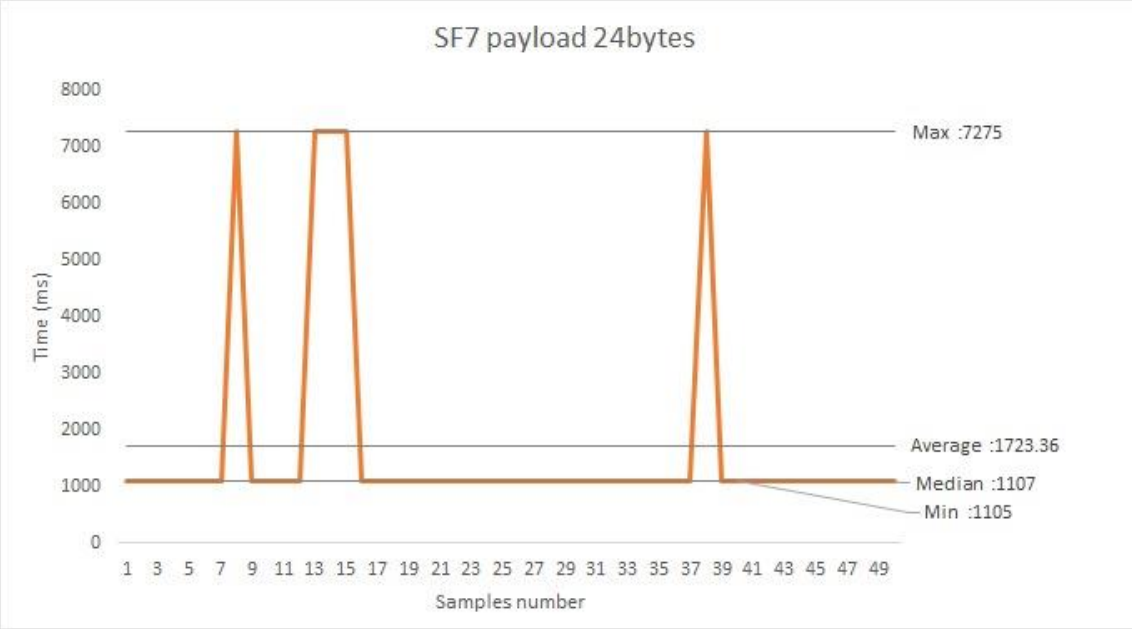

Fig. 4.14 RTT SF = 7 Payload 24 bytes

Some other statistical parameters from the previous mentioned are:

- Mean = 1723.36 ms
- Median = 1107 ms
- Min = 1105 ms
- Standard deviation = 1868.93 ms

SF = 7 Payload = 51 bytes
In this case only a variation on the payload size was made with respect to the previous case. A 51 bytes' payload was selected because LMIC library supports data until 51 bytes and also because in many regions, the maximum payload size is 51 bytes for a SF = 12.
In this case from the samples taken (annex 4) it can be observed that the RTT, for the majority of the samples, has a small increase. In this case there were also retransmissions with a maximum RTT = 21809 ms. In this case the messages that were successfully transmitted once have a RTT that is about 1147 ms to 1149 ms.
In Figure 4.15 a graphical representation of the 50 samples as a summary is shown.

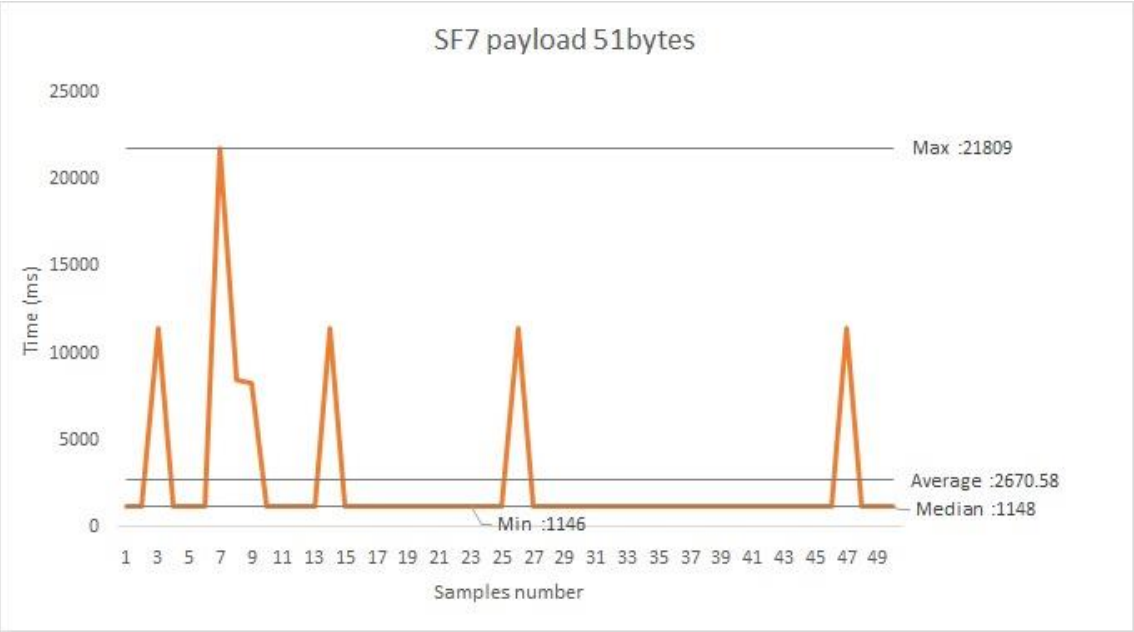

Fig. 4.15 RTT SF = 7 Payload 51 bytes

In order to verify that the SF selected is in reality being used, the traffic from the gateway can be analyzed. In Figure 4.16 it can be seen, under data rate column, that the SF for the uplink is indeed SF = 7.

time	frequency	mod.	CR	data rate	airtime (ms)	cnt				
▼ 16:00:18	868.1	lora	4/5	SF 7 BW 125	41.2	50	dev addr:	26 01 12 F7	payload size:	12 bytes
▲ 16:00:17	868.1	lora	4/5	SF 7 BW 125	102.7	0	dev addr:	26 01 12 F7	payload size:	51 bytes

Fig. 4.16 SF = 7 Verification

Some other statistical parameters from the previous mentioned test are:

- Mean = 2670.58 ms
- Median = 1148 ms
- Min = 1146 ms
- Standard deviation = 4131.66 ms

SF = 9 Payload = 51 bytes

In this case, an intermediate SF has been selected with the same payload from the previous case.

From the samples taken (annex 4) it can be observed that the RTT, for the majority of the samples, has a small increase. In this case there was only one retransmission with a RTT = 55681 ms. In this case the messages that were successfully transmitted once have a RTT that is about 22812 ms to 22816 ms.

In Figure 4.17 a graphical representation of the 50 samples as a summary is shown.

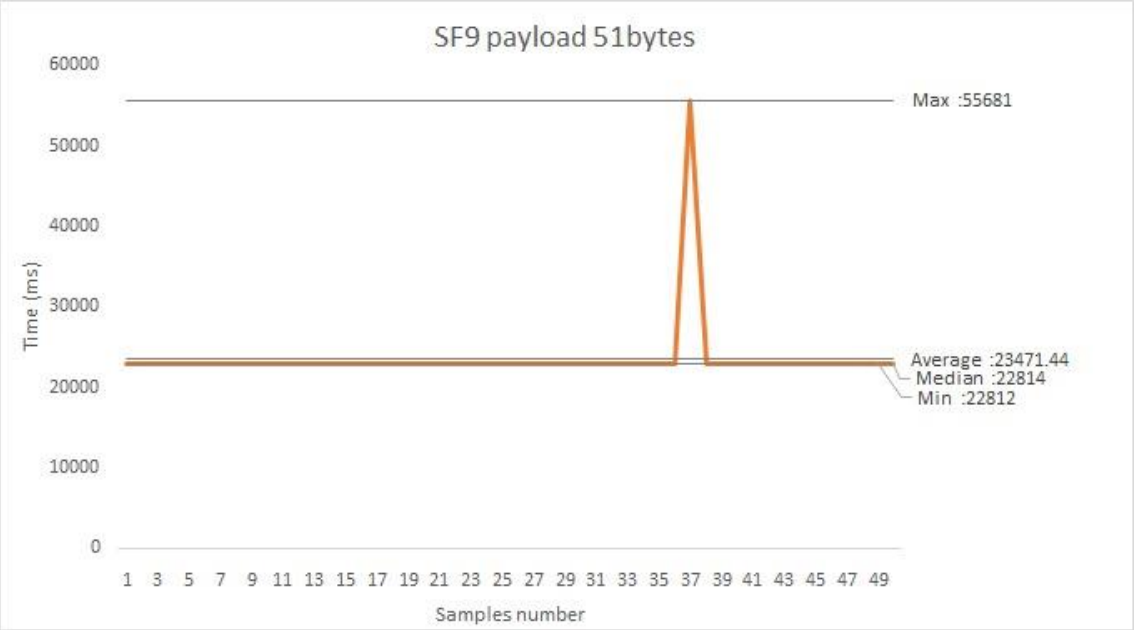

Fig. 4.17 RTT SF = 9 Payload 51 bytes

In order to verify that the SF selected is in reality being used, the traffic from the gateway can be analyzed. In Figure 4.18 it can be seen under data rate column that the SF for the uplink is indeed SF = 9.

time	frequency	mod.	CR	data rate	airtime (ms)	cnt			
▼ 16:23:14	869.525	lora	4/5	SF 9 BW 125	144.4	89	dev addr:	26 01 12 F7	payload size: 12 bytes
▲ 16:23:13	868.1	lora	4/5	SF 9 BW 125	328.7	9	dev addr:	26 01 12 F7	payload size: 51 bytes

Fig. 4.18 SF = 9 Verification

Some other statistical parameters from the previous mentioned are:

- Mean = 23471.44 ms
- Median = 22814 ms
- Min = 22812 ms
- Standard deviation = 4648.08 ms

SF = 12 Payload = 51 bytes

In this case an intermediate SF has been selected with the same payload from the previous case.

In this case from the samples taken (annex 4) it can be observed that the RTT, for the majority of the samples, has a small increase. In this case there was only one retransmissions with a RTT = 483131 ms. In this case the messages that were successfully transmitted once have a RTT that is about 236537 ms to 236541 ms.

In Figure 4.19 a graphical representation of the 50 samples as a summary is shown.

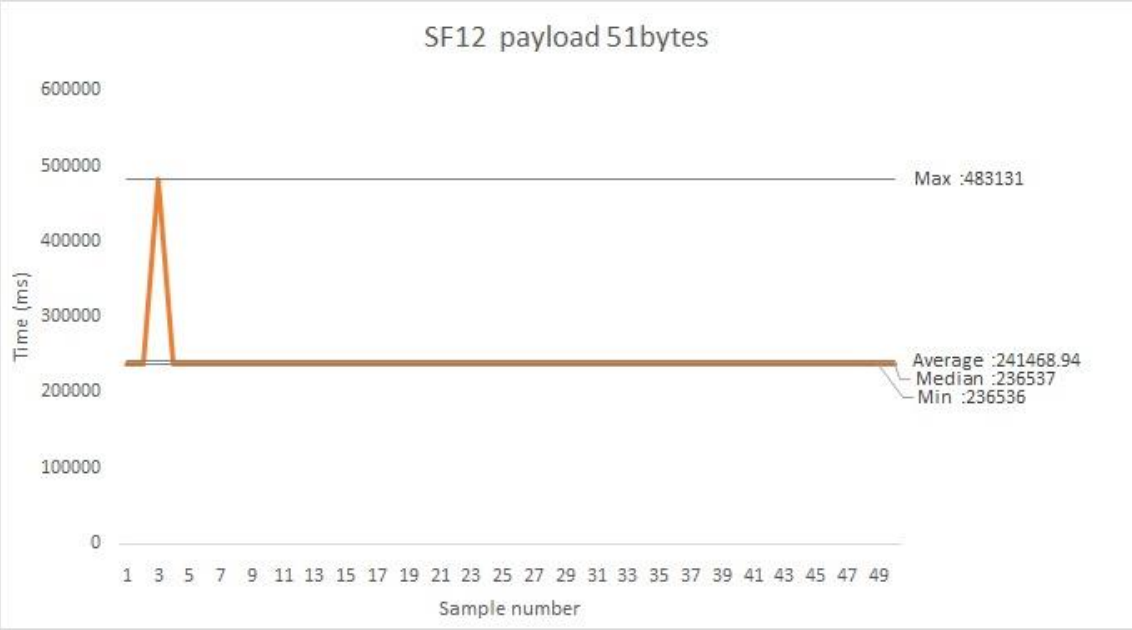

Fig. 4.19 RTT SF = 12 Payload 51 bytes

In order to verify that the SF selected is in reality being used, the traffic from the gateway can be analyzed. In Figure 4.20 it can be seen under data rate column that the SF for the uplink is indeed SF = 12.

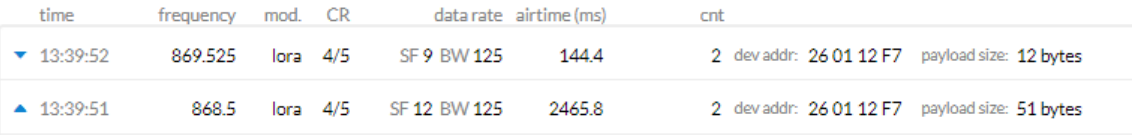

Fig. 4.20 SF = 12 Verification

Some other statistical parameters from the previous mentioned are:

- Mean = 241468.94 ms
- Median = 236537 ms
- Min = 236536 ms
- Standard deviation = 34873.64 ms

Based on the tests ran with different SF and same payload, and different payload but same SF, it can be concluded that changing the SF level from one to another implies a very big change in in terms of the measured RTT. Meanwhile, changing the payload to almost double and maintaining the SF doesn't change the RTT by a lot, it is almost imperceptible.

The high RTT values for SF = 9 and SF = 12 are mainly due to LoRaWAN regulation limits, which state that duty cycle for the band used to transmit the

packages belong to g1, noted previously on section 1.4.6, which is 1%. The measuring of RTT starts when the packet is ready to be sent, but its transmission would happen after the necessary period to meet the regulations.

All the four SF cases presented have to meet the mentioned regulations, but the RTT is very high for SF = 9 and SF = 12 due to the bit rate for each case, which is shown next:

- SF7 bit rate: 5470 bit/s
- SF9 bit rate: 1760 bit/s
- SF12 bit rate: 250 bit/s

The time between 2 consecutive transmissions (meeting the 1% duty cycle regulation) increases with the air time of a packet, which is inversely proportional to the bit rate. For a lower bit rate, a higher air time, thus a high wait time in order to comply with the duty cycle requirements.

In order to obtain the RTT without taking into account the time the device has to wait for, a delay in the transmission interval has been set. The delay is of 30 seconds and 240 seconds for SF9 and SF12, respectively. The following mean from 5 samples in each case has been obtained:

- SF = 9 -- RTT = 2817 ms
- SF = 12 -- RTT = 6540 ms

In Figure 4.21 and 4.22 the outcome for each SF, in the Arduino serial monitor after the delay in the transmission interval, are shown.

```
Sending Packet
Start time:    32541
EV_TXCOMPLETE (includes waiting for RX windows)
End time:    35358
RTT (milliseconds):    2817
Data Received:
Sending Packet
Start time:    65411
EV_TXCOMPLETE (includes waiting for RX windows)
End time:    68227
RTT (milliseconds):    2816
Data Received:
```

Fig. 4.21 SF = 9 With a wait delay of 30 seconds

```
Start time:    244678
EV_TXCOMPLETE (includes waiting for RX windows)
End time:    251219
RTT (milliseconds):    6541
Data Received:
Sending Packet
Start time:    491273
EV_TXCOMPLETE (includes waiting for RX windows)
End time:    497813
RTT (milliseconds):    6540
Data Received:
```

Fig. 4.22 SF = 12 With a wait delay of 240 seconds

From the summary graphs from all the cases it can be seen that with lower SF the retransmissions are more frequent than with higher SF.

A summary of the obtained values for the different SF is shown in table 4.1

Table 4.1. Values Summary for the SF

	SF7 payload 24 bytes	SF7 payload 51 bytes	SF9 payload 51 bytes	SF12 payload 51 bytes
MEAN (ms)	1723.36	2670.58	23471.44	241468.94
MEDIAN (ms)	1107	1148	22814	236537
MIN (ms)	1105	1146	22812	236536
MAX (ms)	7275	21809	55681	483131
STANDARD DEVIATION (ms)	1868.9305	4131.6631	4648.0815	34873.64928

4.2 Educational Use-Case for a LoRaWAN Lab session

In order for a better understanding of the LoRa and LoRaWAN key concepts, the LoRaWAN environment explained in previous sections could be used. A simple and practical scenario for educational purposes is presented next.

4.2.1 Administration Roles

Before setting up all the roles, all of the participants including the professor or the laboratory technician must have a TTN account.

Administrator
The course professor or the laboratory technician should be set as the owner of the gateway or be the only collaborator with full access to all the features (presented in Figure 3.7).

Students
Depending on the goal of the lab session the administrator has to add each student´s account as a collaborator and give the desired permissions. Recommend permissions are:

- **Gateway location:** View the exact location of the gateway
- **Gateway messages:** View the traffic that occurs on the gateway

With these permissions, the student can see the incoming and outgoing traffic that goes through the Gateway and analyse the packets and every parameter that was shown in section 4.1.1.

Each student can create his/her own application or also the professor or the laboratory technician can create a unique application and assign collaborators to it. The second option is recommended when the number of end-devices is limited because each of the student can access the same device by being a collaborator of the application the device is registered to. The permissions that can be managed at the moment of the creation of a collaborator can be seen in Figure 3.8. The recommended permissions are:

- **Devices:** View edit and add devices of the application
- **Settings:** Manage the application settings and access keys

With these permissions the student can register an end-device and analyse uplink and downlink traffic and analyse the packets and every parameter that was shown in section 4.1.2. Furthermore, the student can send downlink messages and integrate other third party APIs.

CONCLUSIONS

In this master thesis, the LPWAN paradigm for the IoT connectivity has been introduced and in a more in-depth explanation LoRa technology has been described, which is a solution based on long-range, low-power consumption, radio links, that can have numerous transmission parameter combinations.

A LoRaWAN environment has been developed with educational purposes where a student can learn the basic LoRa/LoRaWAN concepts in a practical matter. The student can benefit from a step by step guide where various parameters can be managed and analysed such as:

- How to activate a gateway.
- How to activate an end-device.
- How to send Uplink/Downlink packet.
- How to decode a LoRa packet.
- How to analyse a LoRa packet.
- Time and trace analysis.
- Check link parameters and status.
- Analysis of the SF impact

LoRa modulation is a physical layer whose main advantages are: high tolerance to interference thanks to the use of the spread spectrum modulation, long range transmissions in the order of kilometres, very low power consumption, years of lifetime with battery use.

LoRaWAN on the other hand is a network protocol that uses the LoRa modulation to communicate and manage LoRa devices, that are composed of mainly two parts, gateways and end-devices.

Analysing the different network servers that exist, the chosen network server is TTN NS. One of the main reasons for this was that the Gateway used for this project is not compatible for the moment with any server except for The Things Network Server. Apart from this reason, also the TTN server has some other features that its competitors are not yet fulfilling as it is the case of integrations with other services. TTN offers a wide spectrum of integrations with different third party developers.

One of the issues with TTN NS and all the other servers analysed is that setting up a network and understanding all the features that LoRa can be complicated for a person that doesn’t have a solid knowledge of LoRaWAN. The main objective of this thesis helps in this situation due to the fact that besides setting up a practical LoRaWAN scenario it is intended to be taken as a step by step guide for a better understanding of how LoRaWAN technology works.

A practical LoRaWAN scenario was developed in two ways: one using The Things network resources and another one setting up a private environment, the two of them gave similar results because for now TTN is in charge of the account

servers and the gateway has to register in this server in order to start forwarding packets.

Future Works

LoRaWAN is a wide topic and different lines of work can be followed using the present work due to the fact that this master thesis is the basis for building a bigger LoRa network. The gateway is capable of connecting thousands of devices at the same time.

Future works can integrate different APIs to solve all kinds of needs. Also later this year TTN is releasing a new firmware version for the gateway and a new network server where more features are going to be implemented such as running a full private environment.

Another future work item is analysing the security of LoRaWAN links and compare the security level between OTAA and ABP.

Sustainability considerations

LoRaWAN is intended for large networks where a lot of devices are connected to a single gateway within a radius of up to 10 km. For this reason, the end-devices that are spread need to be cheap from an economic point of view. Regarding the energy consumption, the majority of end-devices will run on batteries with very small power consumption, with the battery lasting for years.

However, it must be taken into account the radiofrequency regulations which state that the maximum transmitted power an emitter can use when communicating is 25 mW. And that the maximum ratio of time on the air is 1% per hour.

Ethical Considerations

This project contributes to upcoming students, who can benefit from learning LoRa/LoRaWAN technology in a practical and guided way. Due to the fact that nowadays, and probably in the future, this technology is very relevant in the IoT area.

ACRONYMS

ABP	Activation by Personalization
ADR	Adaptive Data Rate
BW	Bandwidth
CF	Carrier Frequency
CLI	Command Line Interface
CRC	Cyclic Redundancy Check
CSS	Chip Spread Spectrum
IOT	Internet of Things
LPWAN	Low Power Wide Area Network
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LR-WPANs	Low-Rate Wireless Personal Area Networks
MAC	Medium Access Control
NS	Network Server
OSI	Open Systems Interconnection
OTAA	Over the Air Activation
RFID	Radio Frequency ID
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal-to-noise ratio
TP	Transmission Power
TTN	The Things Network
UNB	Ultra Narrow Band
WSN	Wireless Sensor Network

REFERENCES

[1] D. Evans, “The internet of things: How the next evolution of the internet is changing everything,” *CISCO white Pap.*, vol. 1, no. 2011, pp. 1–11, 2011.

[2] L. Vangelista, A. Zanella, and M. Zorzi, “Long-range IoT technologies: The dawn of LoRa,” in *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*, 2015, pp. 51–58.

[3] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, “A study of LoRa: Long range & low power networks for the internet of things,” *Sensors*, vol. 16, no. 9, p. 1466, 2016.

[4] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, “Long-range communications in unlicensed bands: the rising stars in the IoT and smart city scenarios,” *IEEE Wirel. Commun.*, vol. 23, no. 5, pp. 60–67, 2016.

[5] I. 802 W. Group and others, “IEEE Standard for Local and Metropolitan Area Networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs),” *IEEE Std*, vol. 802, pp. 4–2011, 2011.

[6] The Bluetooth Special Interest Group, “Bluetooth 5 Bluetooth Technology Website,” *Technical Spec for Bluetooth 5*, 2018. .

[7] E. Khorov, A. Lyakhov, A. Krotov, and A. Guschin, “A survey on IEEE 802.11 ah: An enabling networking technology for smart cities,” *Comput. Commun.*, vol. 58, pp. 53–69, 2015.

[8] U. Raza, P. Kulkarni, and M. Sooriyabandara, “Low power wide area networks: An overview,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 2, pp. 855–873, 2017.

[9] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent, “Lorawan specification,” *LoRa alliance*, 2015.

[10] M. Bor and U. Roedig, “LoRa transmission parameter selection,” 2017.

[11] Semtech, “LoRa Modulation Basics v2,” 2015.

[12] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke, “A survey of lorawan for iot: From technology to application,” *Sensors*, vol. 18, no. 11, p. 3995, 2018.

[13] T. M. Workgroup, “Technical Overview of LoRa and LoRaWAN,” 2015.

[14] TTN, “LoRaWAN Duty-Cycle.” [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html.

[15] TTN, “TTN LoRaWAN Documentation.” [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/.

[16] Dragino, “LoRa Shield for Arduino.” [Online]. Available:

http://www.dragino.com/products/module/item/102-lora-shield.html.

[17] RSComponents, “TTN Gateway V1.” [Online]. Available: https://uk.rs-online.com/web/p/wifi-routers/1359783/.

[18] TTN, “The Things Gateway.”

[19] T. COSTACHIOIU, “Hands-on: The Things Network Gateway,” *Hands-on: The Things Network Gateway*, 2018. [Online]. Available: https://electronza.com/review-ttn-things-network-gateway/.

[20] LoRaServer, “LoRa Server, open-source.” .

[21] TTN, “The Things Network.”

[22] T. T. Network, “The Things Network Learn,” *Learn LoRaWAN*. [Online]. Available: https://www.thethingsnetwork.org/docs/gateways/.

[23] “MQQT ORG.”

[24] L. Foundation, “Node.js.” [Online]. Available: https://nodejs.org/es/about/.

[25] IBM, “IBM LMIC Framework.” [Online]. Available: https://www.arduinolibraries.info/libraries/ibm-lmic-framework.

[26] H. Visser, “Setting up a Private Routing Environment.”

ANNEXES

ANNEX 1

The following are the steps to create a Private Back-End

First it is necessary to have a working directory created where all the network components are going to be installed in the case of the project the directory is `~/ttn`.

Discovery Server

The first step is to create a configuration file (ttn.yml) under `~/ttn/discovery/ttn.yml`.

Configuration File:

```
id: mynetwork-discovery
tls: true
key-dir: discovery/
auth-servers:
  ttn-account-v2: "https://account.thethingsnetwork.org"
  local: "file://discovery/server.pub"

discovery:
  master-auth-servers:
  - ttn-account-v2
  - local
```

This configuration creates a discovery server called "mynetwork-discovery". The server will use TLS and get its keys from `./discovery`. It uses the public TTN account server and a local certificate for authentication, and both "account servers" are seen as "master servers".

When the configuration file ready now it is necessary to generate public/private keypair for the Discovery server:

```
$ ttn discovery gen-keypair --config ./discovery/ttn.yml
```

Since TLS is used, it is necessary to create a certificate for localhost:

```
$ ttn discovery gen-cert localhost --config ./discovery/ttn.yml
```

To continue with the next steps, the discovery server needs to be running, to start the network server it's necessary to run the following command line:

```
$ ttn discovery --config ./discovery/ttn.yml
```

Access Tokens

In order for the router, broker and handler to authenticate with the discovery server Access tokens are needed, these access tokens have to be created in order to build a private network.

```
ttn discovery authorize router mynetwork-router --config ./discovery/ttn.yml
```

This step has to be replicated in order to get all the access tokens for each part: router, broker, handler and network server.

Router

The second step is to configure a router, for this we have to create another configuration file `~/ttn/router/ttn.yml`.

```
id: mynetwork-router
tls: true
key-dir: router/
auth-servers:
  ttn-account-v2: "https://account.thethingsnetwork.org"

discovery-address: "localhost:1900"
auth-token: THE DISCOVERY ACCESS TOKEN THAT YOU GENERATED FOR THE **ROUTER**

router:
  server-address-announce: localhost
  skip-verify-gateway-token: true
```

This configuration creates a router called "mynetwork-router". The server will use TLS and get its keys from `./router`. It uses the public TTN account server for authentication. It connects to the discovery server on localhost and uses the access token, that was generated in the previous step, to authenticate with the discovery server.

Just like with the discovery server, a public/private keypair and a TLS certificate need to be generated:

```
ttn router gen-keypair --config ./router/ttn.yml
ttn router gen-cert --config ./router/ttn.yml
```

The TLS certificate that was generated for the discovery server is not going to be trusted by the OS and it needs to be appended to a ca.cert file in the router directory:

```
cat discovery/server.cert > router/ca.cert
```

This step has to be replicated when setting up the broker and handler also.

Network Server

The third step is to configure a network, for this the creation of a configuration file is needed `~/ttn/networkserver/ttn.yml`.

```
id: mynetwork-networkserver
tls: true
key-dir: networkserver/
auth-servers:
  ttn-account-v2: "https://account.thethingsnetwork.org"
```

The public/private keypair and TLS certificate have to be generated:

```
ttn networkserver gen-keypair --config ./networkserver/ttn.yml
ttn networkserver gen-cert --config ./networkserver/ttn.yml
```

Broker

Similar to previous steps a configuration file is needed `~/ttn/broker/ttn.yml.`

```
id: mynetwork-broker
tls: true
key-dir: broker/
auth-servers:
  ttn-account-v2: "https://account.thethingsnetwork.org"

discovery-address: "localhost:1900"
auth-token: THE DISCOVERY ACCESS TOKEN THAT YOU GENERATED FOR THE **BROKER**

broker:
  server-address-announce: localhost
  networkserver-cert: broker/networkserver.cert
  networkserver-token: THE NETWORKSERVER ACCESS TOKEN THAT YOU GENERATED
```

The public/private keypair and TLS certificate have to be generated:

```
ttn broker gen-keypair --config ./broker/ttn.yml
ttn broker gen-cert --config ./broker/ttn.yml
```

Now a device address prefix needs to be set, TTN has a reserved prefix for private network which is 26000000/20. This prefix allows to issue 4096 distinct addresses to devices a private network.

```
ttn broker register-prefix 26000000/20 --config ./broker/ttn.yml
```

Handler

The configuration for the Handler will be stored in `~/ttn/handler/ttn.yml.`

```
id: mynetwork-handler
tls: true
key-dir: handler/
auth-servers:
  ttn-account-v2: "https://account.thethingsnetwork.org"

discovery-address: "localhost:1900"
auth-token: THE DISCOVERY ACCESS TOKEN THAT YOU GENERATED FOR THE **HANDLER**

handler:
  server-address-announce: localhost
  broker-id: mynetwork-broker
  mqtt-address: localhost:1883
```

Starting Everything

The final step is to start everything in separate tabs:

```
ttn router --config router/ttn.yml
ttn networkserver --config networkserver/ttn.yml
ttn broker --config broker/ttn.yml
ttn handler --config handler/ttn.yml
gateway-connector-bridge  --root-ca-file "bridge/ca.cert"  --ttn-router
"localhost:1900/mynetwork-router"
```

After everything is started the network can be managed with the line command “ttnctl”.

Annex 2

The following annex is the Arduino code for an end-device with ABP activation, the explanation is giving in a form of comments in the code.

This code is capable of sending uplink messages as well receiving downlink messages coming from TTN that can be seen in the Arduino serial monitor. It can also allow ACKs as it can be observed in

```
// Libraries needed for the code
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// LoRaWAN NwkSKey, obtained from the TTN page in the device section

static const PROGMEM u1_t NWKSKEY[16] = { 0x95, 0xFE, 0xA2, 0x7F, 0x1F, 0xA6,
0x46, 0xE6, 0x45, 0xCC, 0x0A, 0xEB, 0xE7, 0x1F, 0xD0, 0x3D };

// LoRaWAN AppSKey, obtained from the TTN page in the device section

static const u1_t PROGMEM APPSKEY[16] ={ 0xA2, 0xE9, 0xAD, 0x13, 0xF5, 0xBA,
0x03, 0x14, 0x92, 0xAE, 0xE1, 0x6A, 0xCB, 0x35, 0x97, 0x4D };



// LoRaWAN end-device address (DevAddr), obtained from the TTN page in the
device section

static const u4_t DEVADDR = 0x260117DC;

// Time analysis variable declaration
unsigned long startTime;
unsigned long RTT;
unsigned long endTime;

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }
```

```
//Payload to be sent
static uint8_t mydata[] = "22222";
static osjob_t initjob,sendjob,blinkjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 10; //for SF 7
//const unsigned TX_INTERVAL = 30; //for SF 9
//const unsigned TX_INTERVAL = 240; //for SF 12

// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println("OP_TXRXPEND, not sending");
    } else {
// Prepare upstream data transmission at the next possible time by setting
// the 4th parameter to one ACK is required
        LMIC_setTxData2(1, mydata, sizeof(mydata), 0);
        Serial.println("Packet queued");
        Serial.println(LMIC.freq);

         startTime=millis(); // START TIMING PROCESS FOR RTT
    }
    // Next TX is scheduled after TX_COMPLETE event.
}

void onEvent (ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    Serial.println(ev);
    switch(ev) {
        case EV_SCAN_TIMEOUT:
```

```
            Serial.println("EV_SCAN_TIMEOUT");
            break;
        case EV_BEACON_FOUND:
            Serial.println("EV_BEACON_FOUND");
            break;
        case EV_BEACON_MISSED:
            Serial.println("EV_BEACON_MISSED");
            break;
        case EV_BEACON_TRACKED:
            Serial.println("EV_BEACON_TRACKED");
            break;
        case EV_JOINING:
            Serial.println("EV_JOINING");
            break;
        case EV_JOINED:
            Serial.println("EV_JOINED");
            break;
        case EV_RFU1:
            Serial.println("EV_RFU1");
            break;
        case EV_JOIN_FAILED:
            Serial.println("EV_JOIN_FAILED");
            break;
        case EV_REJOIN_FAILED:
            Serial.println("EV_REJOIN_FAILED");
            break;
        case EV_TXCOMPLETE:
            Serial.println("EV_TXCOMPLETE (includes waiting for RX
windows)");
////////////////RTT CALCULATION//////////////////////////////
             endTime=millis(); // END TIMING THE RTT
            RTT=endTime-startTime; // RTT CALCULATION
            Serial.print("RTT:    "); // PRINT THE RTT
            Serial.println(RTT);
//////////////////////////////////////////////////////////////

//////// CODE TO RECEIVE PACKETS FROM TTN/////////////////
            if(LMIC.dataLen) {
                // data received in rx slot after tx
                Serial.print("Data Received: ");
```

```
                Serial.write(LMIC.frame+LMIC.dataBeg, LMIC.dataLen);
                Serial.println();
            }
////////////////////////////////////////////////////////////////
            // Schedule next transmission
            os_setTimedCallback(&sendjob,
os_getTime()+sec2osticks(TX_INTERVAL), do_send);
            break;
        case EV_LOST_TSYNC:
            Serial.println("EV_LOST_TSYNC");
            break;
        case EV_RESET:
            Serial.println("EV_RESET");
            break;
        case EV_RXCOMPLETE:
            // data received in ping slot
            Serial.println("EV_RXCOMPLETE");
            break;
        case EV_LINK_DEAD:
            Serial.println("EV_LINK_DEAD");
            break;
        case EV_LINK_ALIVE:
            Serial.println("EV_LINK_ALIVE");
            break;
         default:
            Serial.println("Unknown event");
            break;
    }
}
void setup() {
    Serial.begin(9600);
    while(!Serial);
    Serial.println("Starting");
    #ifdef VCC_ENABLE
    // For Pinoccio Scout boards
    pinMode(VCC_ENABLE, OUTPUT);
    digitalWrite(VCC_ENABLE, HIGH);
    delay(1000);
    #endif
```

```
    // LMIC init
    os_init();
    // Reset the MAC state. Session and pending data transfers will be
discarded.
    LMIC_reset();
    //LMIC_setClockError(MAX_CLOCK_ERROR * 1/100);
    // Set static session parameters. Instead of dynamically establishing a
session
    // by joining the network, precomputed session parameters are be
provided.
    #ifdef PROGMEM
    // On AVR, these values are stored in flash and only copied to RAM
    // once. Copy them to a temporary buffer here, LMIC_setSession will
    // copy them into a buffer of its own again.
    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
    #else
    // If not running an AVR with PROGMEM, just use the arrays directly
    LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
    #endif

    // Disable link check validation 0 Enabled 1
    LMIC_setLinkCheckMode(0);
    LMIC_setAdrMode(0);
    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;

    // Set data rate and transmit power
    LMIC_setDrTxpow(DR_SF7,14);

    // Start job
    do_send(&sendjob);
}

void loop() {
    os_runloop_once();
}
```

Annex 3

The script for Node.js in order to decode a LoRaWAN packet has to be ran in the CLI:

```
Node >> END

const lorapacket = require('lora-packet');
const nwkSKey = new Buffer('C15C54812EC31DEA211AE337CEC598F2', 'hex');
const appSKey = new Buffer('55B9D78BCF4D70D868DE98749DBFAD09', 'hex');
const data = new Buffer('40F7120126807300011C526EBAF83712C796', 'hex');

const packet = lorapacket.fromWire(data);
console.log(packet.toString());
const valid = lorapacket.verifyMIC(packet, nwkSKey);
console.log('MIC: ' + (valid ? 'OK' : 'FAIL'));

const payload = lorapacket.decrypt(packet, appSKey, nwkSKey);
console.log('Payload: ' + payload.toString());

>>
```

Annex 4

RTT in ms obtained for each case and statistical parameters

	SF7 payload 24 bytes	SF7 payload 51 bytes	SF9 payload 51 bytes	SF12 payload 51 bytes
	1106	1148	22818	236541
	1107	1148	22816	236538
	1107	11410	22816	483131
	1107	1147	22814	236538
	1107	1148	22814	236541
	1108	1148	22814	236538
	1106	21809	22813	236538
	7273	8429	22814	236538
	1108	8304	22815	236537
	1107	1148	22812	236537
	1106	1147	22815	236537
	1107	1147	22814	236537
	7273	1148	22816	236537
	7275	11411	22813	236537
	7273	1147	22814	236537
	1106	1147	22814	236537
	1106	1147	22813	236536
	1107	1149	22814	236537
	1107	1147	22814	236536
	1107	1147	22813	236537
	1106	1148	22815	236536
	1107	1148	22814	236537
	1107	1146	22815	236537
	1107	1148	22814	236537
	1106	1148	22814	236537
	1106	11410	22814	236537
	1107	1147	22814	236537
	1107	1148	22814	236537
	1106	1148	22812	236537
	1106	1147	22814	236537
	1107	1148	22812	236537
	1108	1148	22813	236537
	1106	1147	22814	236537
	1106	1147	22814	236536
	1107	1148	22814	236536
	1107	1147	22813	236537
	1106	1147	55681	236537
	7275	1148	22813	236537
	1107	1148	22812	236537
	1105	1147	22813	236537

	1106	1148	22814	236537
	1107	1148	22816	236536
	1107	1147	22815	236537
	1107	1148	22816	236536
	1106	1148	22812	236537
	1107	1147	22814	236537
	1107	11411	22816	236536
	1107	1148	22814	236537
	1106	1148	22814	236536
	1106	1147	22816	236536
MEAN (ms)	1723.36	2670.58	23471.44	241468.94
MEDIAN (ms)	1107	1148	22814	236537
MIN (ms)	1105	1146	22812	236536
MAX (ms)	7275	21809	55681	483131
STANDARD DEVIATION (ms)	1868.9305	4131.6631	4648.0815	34873.64928